00862.023110.

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:	)	
	:	Examiner: NOT YET ASSIGNED
MAKOTO MIHARA	)	
	:	Group Art Unit: NOT YET ASSIGNED
Application No.: 10/600,843	)	
	:	
Filed: June 23, 2003	)	
	:	
For: METHOD, PROGRAM, AND STORAGE MEDIUM FOR ACQUIRING LOGS	)	
	:	
	)	September 24, 2003

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

SUBMISSION OF PRIORITY DOCUMENTS

Sir:

In support of Applicant's claim for priority under 35 U.S.C. § 119, enclosed are certified copies of the following foreign application:

2002-191128, filed June 28, 2002; and

2002-191129, filed June 28, 2002.

Applicant's undersigned attorney may be reached in our New York office by telephone at (212) 218-2100. All correspondence should continue to be directed to our address given below.

Respectfully submitted,

Attorney for Applicant

Registration No. 42,476

FITZPATRICK, CELLA, HARPER & SCINTO
30 Rockefeller Plaza
New York, New York 10112-3801
Facsimile: (212) 218-2200

NY_MAIN 377077v1

CEM03110US
Appln. No. 10/600,843
GAU: 1794

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日
Date of Application: ２００２年 ６月２８日

出 願 番 号
Application Number: 特願２００２－１９１１２８

[ST. 10/C]: [ＪＰ２００２－１９１１２８]

出 願 人
Applicant(s): キヤノン株式会社

２００３年 ７月１０日

特許庁長官
Commissioner,
Japan Patent Office 太田信一郎

出証番号 出証特２００３－３０５５５９６

【書類名】　特許願
【整理番号】　4741003
【提出日】　平成14年 6月28日
【あて先】　特許庁長官殿
【国際特許分類】　G06F　9/00
【発明の名称】　ログ取得方法およびプログラム、記憶媒体
【請求項の数】　14
【発明者】
【住所又は居所】　東京都大田区下丸子３丁目３０番２号　キヤノン株式会社内
【氏名】　三原　誠
【特許出願人】
【識別番号】　000001007
【氏名又は名称】　キヤノン株式会社
【代理人】
【識別番号】　100076428
【弁理士】
【氏名又は名称】　大塚　康徳
【電話番号】　03-5276-3241
【選任した代理人】
【識別番号】　100112508
【弁理士】
【氏名又は名称】　高柳　司郎
【電話番号】　03-5276-3241
【選任した代理人】
【識別番号】　100115071
【弁理士】
【氏名又は名称】　大塚　康弘
【電話番号】　03-5276-3241

【選任した代理人】
　【識別番号】　100116894
　【弁理士】
　【氏名又は名称】　木村　秀二
　【電話番号】　03-5276-3241
【手数料の表示】
　【予納台帳番号】　003458
　【納付金額】　21,000円
【提出物件の目録】
　【物件名】　明細書　1
　【物件名】　図面　1
　【物件名】　要約書　1
　【包括委任状番号】　0102485
【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　ログ取得方法およびプログラム、記憶媒体

【特許請求の範囲】

【請求項１】　所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、ポインタ・パラメータが所定の定義方法で定義されているか否かを判断する工程と、

ポインタ・パラメータが所定の定義方法で定義されていた場合、該定義方法に基づいて該ポインタ・パラメータの指すメモリの内容をログとして記録する工程と

を備えることを特徴とするログ取得方法。

【請求項２】　前記ログ取得のための関数は、前記定義方法に基づいてメモリサイズを算出する工程を更に備え、前記記録する工程は、前記ポインタ・パラメータの指すメモリの内容を、該算出したメモリサイズ分、記録することを特徴とする請求項１に記載のログ取得方法。

【請求項３】　所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数を呼び出す際の所定の情報と、前記実行結果を受け取った際の所定の情報とを記録する工程と、

前記プログラム内の関数定義に基づいて、エクスポートされていない関数の有無を判断する工程と、

エクスポートされていない関数があった場合、該関数に対応するログ取得のための関数を新たに生成し、該エクスポートされていない関数のアドレスを、該新たに生成された関数のアドレスに書き換える工程と、を備え、

前記新たに生成された関数は、

前記エクスポートされていない関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記エクスポートされていない関数を呼び出す際の所定の情報と、前記実行結果を受け取った際の所定の情報とを記録する工程と

を備えることを特徴とするログ取得方法。

【請求項４】　所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、可変長配列としてポインタ・パラメータが定義されているか否かを判断する工程と、

可変長配列としてポインタ・パラメータが定義されていた場合、該ポインタ・パラメータの指すメモリの内容を該指定された配列で記録する工程と

を備えることを特徴とするログ取得方法。

【請求項５】　所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った

実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義に、構造体としてポインタ・パラメータが指定されているか否かを判断する工程と、

構造体としてポインタ・パラメータが指定されていた場合、該ポインタ・パラメータの指すメモリの内容について、前記プログラム内の関数定義において定義された構造体のサイズ分ログとして記録する工程と

を備えることを特徴とするログ取得方法。

【請求項６】　所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、構造体として種類指定がなされたポインタ・パラメータが定義されているか否かを判断する工程と、

構造体として種類指定がなされたポインタ・パラメータが定義されていた場合、該ポインタ・パラメータの指すメモリの内容を、種類指定がなされたデータ型として記録する工程と

を備えることを特徴とするログ取得方法。

【請求項７】　所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、ポインタ・パラメータが所定の定義方法で定義されているか否かを判断する工程と、

ポインタ・パラメータが所定の定義方法で定義されていた場合、該定義方法に基づいて該ポインタ・パラメータの指すメモリの内容をログとして記録する工程と

を備えることを特徴とするログ取得方法。

【請求項８】　前記ログ取得のためのメソッドは、前記定義方法に基づいてメモリサイズを算出する工程を更に備え、前記記録する工程は、前記ポインタ・パラメータの指すメモリの内容を、該算出したメモリサイズ分、記録することを特徴とする請求項７に記載のログ取得方法。

【請求項９】　所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行うメソッドを呼び出す際の所定の情報と、前記実行結果を受け取った際の所定の情報とを記録する工程と、

前記プログラム内の関数定義に基づいて、エクスポートされていないメソッドの有無を判断する工程と、

エクスポートされていないメソッドがあった場合、該メソッドに対応するログ取得のためのメソッドを新たに生成し、該エクスポートされていないメソッドのアドレスを、該新たに生成されたメソッドのアドレスに書き換える工程と、を備え、

前記新たに生成されたメソッドは、

前記エクスポートされていないメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記エクスポートされていないメソッドを呼び出す際の所定の情報と、前記実行結果を受け取った際の所定の情報とを記録する工程と

を備えることを特徴とするログ取得方法。

【請求項１０】　所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、可変長配列としてポインタ・パラメータが定義されているか否かを判断する工程と、

可変長配列としてポインタ・パラメータが定義されていた場合、該ポインタ・パラメータの指すメモリの内容を該指定された配列で記録する工程と

を備えることを特徴とするログ取得方法。

【請求項１１】　所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義に、構造体としてポインタ・パラメータが指定されているか否かを判断する工程と、

構造体としてポインタ・パラメータが指定されていた場合、該ポインタ・パラメータの指すメモリの内容について、前記プログラム内の関数定義において定義された構造体のサイズ分ログとして記録する工程と

を備えることを特徴とするログ取得方法。

【請求項１２】　所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、構造体として種類指定がなされたポインタ・パラメータが定義されているか否かを判断する工程と、

構造体として種類指定がなされたポインタ・パラメータが定義されていた場合、該ポインタ・パラメータの指すメモリの内容を、種類指定がなされたデータ型として記録する工程と

を備えることを特徴とするログ取得方法。

【請求項１３】　請求項１乃至１２のいずれか１つに記載のログ取得方法をコンピュータによって実現させるための制御プログラムを格納した記憶媒体。

【請求項１４】　請求項１乃至１２のいずれか１つに記載のログ取得方法をコンピュータによって実現させるための制御プログラム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、複数にモジュール分けされたソフトウェアの処理ログを取得するための技術に関するものである。

【０００２】

【従来の技術】

従来より、再現率の低いソフトウェアの障害に対しては、ソフトウェアの処理ログを取得し、当該処理ログを解析することによって、障害の原因をつきとめ、その対策を講じてきた。

【０００３】

【発明が解決しようとする課題】

しかし、上記従来の処理ログの取得には以下のような問題点がある。

（１）ユーザの動作環境でもログを取得しつづけるためには、ソフトウェアのモジュール自体に手を加え、処理ログ取得ルーチンを追加しなければならず、処理ログ取得のための作業負荷が大きかった。

（２）処理ログ取得はモジュール毎に行うため、生成されたログはモジュール単位のものとなってしまい、ソフトウェア全体の処理を、完全に時間順のログとして取得するのが困難である。このため、全体の処理ログとしての見通しが悪く、ログを解析して障害の原因を発見するまでのプロセスに工数がかかっていた。

【０００４】

本発明は、上記課題を鑑みてなされたものであり、複数にモジュール分けされたソフトウェアの処理ログを容易に取得でき、かつ、ソフトウェアの障害の原因の解析のための工数を削減することが可能なログ取得方法、ならびに該方法をコンピュータによって実現させるためのプログラム、および該プログラムを格納した記憶媒体を提供することを目的とする。

【０００５】

【課題を解決するための手段】

上記の目的を達成するために本発明に係るログ取得方法は以下のような構成を備える。即ち、

所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記プログラム内の関数定義において、ポインタ・パラメータが所定の定義方法で定義されているか否かを判断する工程と、

ポインタ・パラメータが所定の定義方法で定義されていた場合、該定義方法に基づいて該ポインタ・パラメータの指すメモリの内容をログとして記録する工程とを備える。

【０００６】

【発明の実施の形態】

【第１の実施形態】

本実施形態は、あるモジュールから別のモジュール内に存在する関数の呼び出しが行われる際の仕組みである、メモリに保持されたインポート関数アドレス、または仮想関数アドレステーブル（Ｖｉｒｔｕａｌ　Ａｄｄｒｅｓｓ　Ｔａｂｌｅ）を利用して、モジュール間の関数呼び出しをフックしてログに記録することで、ソフトウェアのモジュール自体に手を加えることなく、ソフトウエア全体の処理を、時間順のログとして取得することを可能にするものである。以下に具体的に説明する。

【０００７】

＜システム構成＞

図１は、本発明の各実施形態にかかるログ取得方法を実現するコンピュータ（ソフトウェア評価システム）の構成をあらわす図である。説明を簡略化するために、本実施形態では、本ソフトウェア評価システムが１台のＰＣ内部に構築されるものとするが、本発明のログ取得方法の特徴は１台のＰＣ内部に構築されるか、あるいは複数のＰＣにネットワークシステムとして構築されるかによらず有効である。

【０００８】

本ソフトウェア評価システムを搭載するコンピュータには、ＣＰＵ１、チップセット２、ＲＡＭ３、ハードディスクコントローラ４、ディスプレイコントローラ５、ハードディスクドライブ６、ＣＤ－ＲＯＭドライブ７、ディスプレイ８が搭載されている。また、ＣＰＵとチップセットを繋ぐ信号線１１、チップセット２とＲＡＭ３とを繋ぐ信号線１２、チップセット２と各種周辺機器とを繋ぐ周辺機器バス１３、ハードディスクコントローラ４とハードディスクドライブ６とを繋ぐ信号線１４、ハードディスクコントローラ４とＣＤ－ＲＯＭドライブ７とを繋ぐ信号線１５、ディスプレイコントローラ５とディスプレイ８とを繋ぐ信号線１６が搭載されている。

【０００９】

＜関数処理に対する処理ログ取得＞

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムを説明するために、まず図２によって、複数のモジュールに分かれたソフ

トウェアが、通常の状態でどのようにメモリにロードされるかを説明する。

【００１０】

通常、複数のモジュールに分かれたソフトウェアは、全体の制御を行う実行ファイルＥＸＥ(２３)と、モジュールとして存在しＥＸＥの補完的な役割を担うダイナミックリンクライブラリＤＬＬ(２７)とに分かれており、メモリにはＥＸＥとＤＬＬの両方がロードされる。ＥＸＥはコードセグメント(２８)とデータセグメント(２９)、そしてインポート関数アドレステーブル(２２)から成っている。更に、インポート関数アドレステーブルは、関数の所属するＤＬＬによって分かれており(２１，　２４)、ＤＬＬごとにそれぞれの関数がロードされたアドレスが書かれている(３０～３５)。ＤＬＬの関数の実体は、ＤＬＬごとに分かれて(２５，　２６)ロードされ、それぞれの関数は該当するＤＬＬの一部としてロードされる(３６～４１)。この図では、１本のＥＸＥがA.ＤＬＬ及びB.ＤＬＬの２つのダイナミックリンクライブラリ内の関数を使用している例を示しており、実際に使用される関数はＦｕｎｃ　ＡＡ，　Ｆｕｎｃ　ＡＢ，　Ｆｕｎｃ　ＡＣ，　Ｆｕｎｃ　ＢＡ，　Ｆｕｎｃ　ＢＢ，　Ｆｕｎｃ　ＢＣの６個となっている。

【００１１】

ＥＸＥのコードセグメント内にあるコードが関数Ｆｕｎｃ　ＡＡを呼び出す場合には、まずインポート関数アドレステーブル内に書かれたＦｕｎｃ　ＡＡのアドレス(３０)が読み込まれる。ここには実際にはA.ＤＬＬの一部として読み込まれたＦｕｎｃ　ＡＡコード(３６)のアドレスが書かれており、そのアドレスをコールすることによって、ＥＸＥのコードはA.ＤＬＬのＦｕｎｃ　ＡＡを呼び出すことができる。

【００１２】

図３は、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図２とは、ログ取得用のコードに対してＩＡＴ　Ｐａｔｃｈ(Ｉｍｐｏｒｔ　Ａｄｄｒｅｓｓ　Ｔａｂｌｅ　Ｐａｔｃｈ)という手法を用いて、関数呼び出しをリダイレクトしているという点で異なっている。

【００１３】

ログ取得が開始されると、メモリ内にはＩＡＴ　Ｐａｔｃｈ用のＤＬＬであるＣ.ＤＬＬ(５８)がロードされる。Ｃ.ＤＬＬはインポート関数アドレステーブル(５２)内に書かれた関数のアドレスを、Ｃ.ＤＬＬ内のログ取得コードであるＦｕｎｃ　ＣＡＡ，Ｆｕｎｃ　ＣＡＢ，Ｆｕｎｃ　ＣＡＣ，Ｆｕｎｃ　ＣＢＡ，Ｆｕｎｃ　ＣＢＢ，Ｆｕｎｃ　ＣＢＣのアドレスに書き換える(６１～６６)。Ｃ.ＤＬＬ内のＦｕｎｃ　ＣＡＡ，Ｆｕｎｃ　ＣＡＢ，Ｆｕｎｃ　ＣＡＣ，Ｆｕｎｃ　ＣＢＡ，Ｆｕｎｃ　ＣＢＢ，Ｆｕｎｃ　ＣＢＣのコード(７３～７８)は、ログを記録すると共に、元々の関数呼び出しを受けるべくメモリにロードされている、該当する関数であるＦｕｎｃ　ＡＡ，Ｆｕｎｃ　ＡＢ，Ｆｕｎｃ　ＡＣ，Ｆｕｎｃ　ＢＡ，Ｆｕｎｃ　ＢＢ，Ｆｕｎｃ　ＢＣ(６７～７２)を呼び出す。

【００１４】

図４Ａは、図３におけるＩＡＴ　Ｐａｔｃｈの処理をあらわす図、図４Ｂはログ取得処理の流れを示すフローチャートである。説明を簡略化するために、この図ではＥＸＥがＡ.ＤＬＬ内のＦｕｎｃ　ＡＡを呼び出す際に、ＩＡＴ　Ｐａｔｃｈによるログ取得コードがどのように動作するかの例をあらわしている。

【００１５】

ＥＸＥ(図４Ａの９１)がＦｕｎｃ　ＡＡをコールすると(図４Ａの９４)、Ｃ.ＤＬＬ内にあるログ取得コードがＤＬＬ名／関数名をメモリに保存し（図４ＢのステップＳ４０２）、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、別メモリに保存する(図４Ａの９５、図４ＢのステップＳ４０３)。その後Ｃ.ＤＬＬは本来呼び出されるはずであった、Ａ.ＤＬＬ(図４Ａの９３)内のＦｕｎｃ　ＡＡをコールする(図４Ａの９６、図４ＢのステップＳ４０４)。Ａ.ＤＬＬのＦｕｎｃ　ＡＡ処理(図４Ａの９７)が終了し、Ｃ.ＤＬＬに制御がリターンすると(図４Ａの９８)、Ｃ.ＤＬＬはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、別メモリに保存する(図４Ａの９９)。その後、Ｃ.ＤＬＬは保存したログ情報をファイルに書き込み(図４Ａの１００、図４ＢのステップＳ４０５)、あたかもＡ.ＤＬＬのＦ

ｕｎｃ　ＡＡが通常通りに終了したかのように、ＥＸＥにリターンする（１０１）。

【００１６】

図５は、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの内部構成をあらわす図である。通常は実行形式のＥＸＥ（１１３）が、ＤＬＬ－１（１１６）やＤＬＬ－２（１１７）内の関数を呼び出すが、ここではＡＰＩトレーサと呼ばれるログ取得コードを埋め込み（１１４）、処理ログを生成している（１１５）。ＡＰＩトレーサは、ＤＬＬ－１やＤＬＬ－２の関数定義を記述したファイル（１１１）と、どのＤＬＬのどの関数のインポート関数テーブルを書き換えてログを取得するかの設定シナリオ（トレースシナリオ１１２）を元に動作する。

【００１７】

＜メソッド処理に対する処理ログ取得＞

次に、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、実行ファイルＥＸＥ（１１８）がＣＯＭ（Ｃｏｍｐｏｎｅｎｔ　Ｏｂｊｅｃｔ　Ｍｏｄｅｌ）サーバでエクスポートされているインターフェースのインスタンス作成時に、どのようにメモリにロードされるかを説明するために、まず、図６によって、通常の状態でどのようにメモリにロードされるかを説明する。

【００１８】

通常、インターフェースのインスタンス作成を行うと、ＣＯＭサーバ内で、要求されたインターフェース（１２１，　１２２）と、そのメソッド（：オブジェクト指向プログラミングにおいて、オブジェクトの実行する手続きを記述したプログラム、１３０～１３５）が作成され、それらは、メモリ上に両方がロードされる。ここで、ｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅ（仮想アドレステーブル）は作成された各インターフェース毎に作られ（１１８，　１２０）、作成要求を行ったＥＸＥに渡される。このｖｉｒｔｕａｌ　ａｄｄｒｅｓ　ｔａｂｌｅには各メソッドの作成されたアドレスが書かれている（１２４～１２９）。ＥＸＥはこれら情報を利用し、各インターフェースに対して呼び出しを行う。この図では

、１本のＥＸＥがＩｎｔｅｒｆａｃｅ　Ａ及びＩｎｔｅｒｆａｃｅ　Ｂの２つのインターフェースのインスタンスを作成しており、そのインターフェース内部のメソッドを使用している例を示しており、実際に使用されているメソッドは、Ｍｅｔｈｏｄ　ＡＡ，Ｍｅｔｈｏｄ　ＡＢ，Ｍｅｔｈｏｄ　ＡＣ，Ｍｅｔｈｏｄ　ＢＡ，Ｍｅｔｈｏｄ　ＢＢ，Ｍｅｔｈｏｄ　ＢＣとなっている。

【００１９】

ＥＸＥのコードが関数Ｍｅｔｈｏｄ　ＡＡを呼び出す場合には、まずｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅ内に書かれたＭｅｔｈｏｄ　ＡＡのアドレス(１２４)が読み込まれる。ここには実際にはＣＯＭサーバのＩｎｔｅｒｆａｃｅ　Ａの一部として作成されたＭｅｔｈｏｄ　ＡＡコード(１３０)のアドレスが書かれており、そのアドレスをコールすることによって、ＥＸＥのコードはＩｎｔｅｒｆａｃｅ　ＡのＭｅｔｈｏｄ　ＡＡを呼び出すことができる。

【００２０】

図７は、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図６とは、ログ取得用のコードに対してＶＴａｂｌｅ　Ｐａｔｃｈ(ｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅ　Ｐａｔｃｈ)という手法を用いて、メソッド呼び出しをリダイレクトしているという点で異なっている。

【００２１】

ログ取得が開始されると、メモリ内にはＶＴａｂｌｅ　Ｐａｔｃｈ用のＤＬＬ(１４３)がロードされる。このＤＬＬはｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅ(１３６，　１３８)内に書かれたメソッドのアドレスを、ＤＬＬ内のログ取得コードであるＭｅｔｈｏｄ　Ａ’Ａ，Ｍｅｔｈｏｄ　Ａ’Ｂ，Ｍｅｔｈｏｄ　Ａ’Ｃ，Ｍｅｔｈｏｄ　Ｂ’Ａ，Ｍｅｔｈｏｄ　Ｂ’Ｂ，Ｍｅｔｈｏｄ　Ｂ’Ｃのアドレスに書き換える(１４５～１５０)。ＤＬＬ内のＭｅｔｈｏｄ　Ａ’Ａ，Ｍｅｔｈｏｄ　Ａ’Ｂ，Ｍｅｔｈｏｄ　Ａ’Ｃ，Ｍｅｔｈｏｄ　Ｂ’Ａ，Ｍｅｔｈｏｄ　Ｂ’Ｂ，Ｍｅｔｈｏｄ　Ｂ’Ｃのコード(１５７～１６２)は、ログを記録すると共に、元々のメソッド呼び出しを受けるべくメモリにロードされている、該当するメソッドであるＭｅｔｈｏｄ　ＡＡ，Ｍｅｔｈｏｄ　ＡＢ，Ｍｅ

ｔｈｏｄ　ＡＣ，　Ｍｅｔｈｏｄ　ＢＡ，　Ｍｅｔｈｏｄ　ＢＢ，　Ｍｅｔｈｏｄ　ＢＣ(１５７～１６２)を呼び出す。

【００２２】

図８Ａは、図７におけるＶＴａｂｌｅ　Ｐａｔｃｈの処理をあらわす図、図８Ｂはログ取得処理の流れを示すフローチャートである。説明を簡略化するために、この図ではＥＸＥがＣＯＭサーバ内のＩｎｔｅｒｆａｃｅ　ＡのＭｅｔｈｏｄ　ＡＡを呼び出す際に、ＶＴａｂｌｅ　Ｐａｔｃｈによるログ取得コードがどのように動作するかの例をあらわしている。

【００２３】

ＥＸＥ(図８Ａの１６３)がＭｅｔｈｏｄ　ＡＡをコールすると(図８Ａの１６６)、ＤＬＬ内にあるログ取得コードがモジュール名／インターフェース名／メソッド名をメモリに保存し（図８Ｂのステップ Ｓ８０２）、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、別メモリに保存する(図８Ａの１６７、図８ＢのステップＳ８０３））。その後ＤＬＬは本来呼び出されるはずであった、ＣＯＭサーバ(図８Ａの１６５)内のＭｅｔｈｏｄ　ＡＡをコールする(図８Ａの１６８、図８ＢのステップＳ８０４））。ＣＯＭサーバのＭｅｔｈｏｄ　ＡＡ処理(図８Ａの１６９)が終了し、ＤＬＬに制御がリターンすると(図８Ａの１７０)、ＤＬＬはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、別メモリに保存する(図８Ａの１７１)。その後、ＤＬＬは保存したログ情報をファイルに書き込み(図８Ａの１７２、図８ＢのステップＳ８０５)、あたかもＣＯＭサーバのＭｅｔｈｏｄ　ＡＡが通常通りに終了したかのように、ＥＸＥにリターンする(図８Ａの１７３)。

【００２４】

図９は、本発明の第１の実施形態にかかるソフトウェア評価システムの内部構成をあらわす図である。通常は実行形式のＥＸＥ(１７６)が、ＣＯＭサーバ１(１７９)やＣＯＭサーバ２(１８０)内のメソッドを呼び出すが、ここではＡＰＩトレーサと呼ばれるログ取得コードを埋め込み(１７７)、処理ログを生成している(１７８)。ＡＰＩトレーサは、ＣＯＭサーバ１(１７９)やＣＯＭサーバ２の関

数定義を記述したファイル(１７４)と、どのＣＯＭサーバのどのインターフェースのどのメソッドのｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅを書き換えてログを取得するかの設定シナリオ(１７５)を元に動作する。

【００２５】

以上の説明から明らかなように、本実施形態にかかるログ取得方法によれば、複数にモジュール分けされたソフトウエアの処理ログの取得において、モジュール自体に変更を加えることなく、モジュール内に用意された関数／メソッドの呼び出しをログとして記録することが可能となり、処理ログ取得のための作業負荷を低減することが可能となる。また、生成されるログは、時間順のログとして取得することができ、ログの解析が容易となるため、ソフトウェアの障害の原因の解析のための工数を削減することが可能となる。

【００２６】

【第２の実施形態】

本実施形態では、通常では、取得しきれないポインタ・パラメータのデータをバイナリとしてログ取得する場合について述べる。

【００２７】

図１０は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義の一例であり、一般的に広く用いられているＩＤＬにより、記述されている。本実施形態にかかるログ取得方法を実現するソフトウェア評価システムに於いては、このＩＤＬをトークン化したタイプライブラリファイルを関数定義ファイルとして使用する。

【００２８】

図１１は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて、ポインタ・パラメータに対してバイナリ取得を指定することで、ポインタ・パラメータのデータの実体をログとして取得する為のＩＤＬによる記述である。

【００２９】

FuncBinidId関数の定義に於いて、long* lplParamに対してcustum(PAT_PARAM_ATTR_ID, "binid_is()")と宣言している(２０１)。ここで、PAT_PARAM_ATTR_ID(

２００)はＩＤＬの中で本ソフトウェア評価システムが利用するための識別子である。ここで、"binid_is()"と定義することにより、このパラメータが指すポインタからパラメータのデータ型サイズ(ここではlong型)分のデータを、バイナリデータとしてログに保存する。

【００３０】

FuncSizeIs関数の定義に於いて、int* lplParamに対してcustum(PAT_PARAM_ATTR_ID, "size_is(dwCount)")と宣言している(２０２)。ここで、"size_is(dwCount)"と定義することにより、このパラメータは第一パラメータであるdwCount個分のデータが有ると扱われる。そして、このパラメータが指すポインタから、(dwCount×パラメータのデータ型サイズ(ここでは、int型))分のデータをバイナリデータとしてログに保存する。

【００３１】

FuncLengthIs関数の定義に於いて、char* lpszParamに対してcustum(PAT_PARAM_ATTR_ID, "length_is(dwLength)")と宣言している(２０３)。ここで、"length_is(dwLength)"と定義することにより、このパラメータは第一パラメータであるdwLength個分のデータが有ると扱われる。そして、このパラメータが指すポインタから、(dwLength×パラメータのデータ型サイズ(ここではchar型))分のデータをバイナリデータとしてログに保存する。

【００３２】

FuncBytesIs関数の定義に於いて、void* lpParamに対してcustum(PAT_PARAM_ATTR_ID, "bytes_is(dwSize)")と宣言している(２０４)。ここで、"bytes_is(dwSize)"と定義することにより、このパラメータは第一パラメータであるdwSizesバイト分のデータが有ると扱われる。そして、このパラメータが指すポインタから、dwSizeバイト分のデータをバイナリデータとしてログに保存する。

【００３３】

FuncBytesIs1関数の定義に於いて、void* lpParamに対してcustum(PAT_PARAM_ATTR_ID, "bytes_is(12)")と宣言している(２０５)。ここで、"bytes_is(12)"と定義することにより、このパラメータは指定された１２バイト分のデータが有ると扱われる。そして、このパラメータが指すポインタから、１２バイト分のデー

タをバイナリデータとしてログに保存する。

【００３４】

図１２は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１１の様に関数定義されている時のログを取得する場合の流れを示す図である。

【００３５】

処理が開始されると（ステップＳ１）、ログ取得を開始し、モジュール名、インターフェース名、関数／メソッド名をＨＤＤに保存する(ステップＳ２)。次にログ取得コードは、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する(ステップＳ３)。そして、関数定義にバイナリ取得の設定が存在するか判別し(ステップＳ４)、存在する場合には、保存するメモリサイズを各定義方法(binid_is, size_is, length_is, bytes_is)毎に計算し(ステップＳ５)、ポインタ・パラメータの指すメモリの内容を算出したサイズ分ＨＤＤに保存する(ステップＳ６)。続いて、元の関数を呼び出す(ステップＳ７)。関数からリターンすると、ログ取得コードはリターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する(ステップＳ８)。そして、関数定義にバイナリ取得の設定が存在するか判別し(ステップＳ９)、存在する場合には、保存するメモリサイズを各定義方法(binid_is, size_is, length_is, bytes_is)毎に計算し(ステップＳ１０)、ポインタ・パラメータの指すメモリの内容を算出したサイズ分ＨＤＤに保存する(ステップＳ１１)。この処理はユーザから終了の命令があると(ステップＳ１２)終了される。

【００３６】

図１３は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１１の定義により取得されたログデータの図である。

【００３７】

関数定義により、バイナリ取得の設定がない場合に取得できるログ(２１０)は、DataIDを除いた部分である。そして、バイナリ取得の設定を行った場合には、DataID及び、バイナリデータのログ(２１１)が取得できる。例えばFuncSizeIsのログを比較すると、バイナリ取得の設定がない場合には、ポインタ・パラメータ

lpParamが指し示すデータをint型で１個分だけは取得可能である。しかし、実際にはこのデータは１０個のint型データの配列を指し示す。そのためバイナリ取得の設定を行った場合には、int型データのサイズである４バイト×１０個分、計４０バイトのデータを取得可能となる。

【００３８】

以上の説明から明らかなように、本実施形態によれば、サイズとポインタ・パラメータを関連付けることにより、通常では、取得しきれないポインタ・パラメータのデータをバイナリとしてログ取得可能となる。

【００３９】

【第３の実施形態】

本実施形態では、コールバック関数等のエクスポートされていない関数をログとして取得する場合について述べる。

【００４０】

図１４は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いてコールバック関数等のエクスポートされていない関数をログとして取得するためのＩＤＬによる記述である。

【００４１】

FuncSetCallBack関数の定義に於いて、DWORD pfnFuncCallBackに対してcustom(PAT_PARAM_ATTR_ID, "funcname_is(FuncCallBack)")と宣言している(２２１)。FuncSetCallBack関数は、モジュールに対してコールバック関数を設定する関数であり、DWORD pfnFuncCallBackは、そのコールバック関数のアドレスを設定するためのパラメータである。ここで、"funcname_is(FuncCallBack)"と定義することにより、ログ取得処理は、このパラメータに渡された値をFuncCallBack関数(２２２)のアドレスとして認識し、ログ取得処理のアドレスに置換する。また、ログ取得処理時に本来のコールバック関数を呼ぶためにオリジナルの値を保存する。これにより、エクスポートされていないコールバック関数のログが取得可能となる。

【００４２】

GetFuncPointer関数の定義に於いて、DWORD pfnFuncInternalに対してcustom(

PAT_PARAM_ATTR_ID, "funcname_is(FuncInternal")と宣言している(２２３)。GetFuncPointer関数は、モジュール内部のエクスポートされていない関数のポインタを取得する関数であり、DWORD pfnFuncInternalは、その非エクスポート関数のポインタをを取得するためのパラメータである。ここで、"funcname_is(FuncInternal)"と定義することにより、ログ取得処理は、このパラメータに渡された値をFuncInternal関数(２２４)のアドレスとして認識し、ログ取得処理のアドレスに置換する。また、ログ取得処理時に本来の非エクスポート関数を呼ぶためにオリジナルの値を保存する。これにより、エクスポートされていないモジュール内部の関数のログが取得可能となる。

【００４３】

GetFuncPointeArray関数(２２５)は、モジュール内部のエクスポートされていない関数群のアドレスをFUNCPOINTERARRAY構造体(２２０)に取得するための関数である。ここで、FUNCPOINTERARRAYの各メンバの定義に対してcustom(PAT_PARAM_ATTR_ID, "funcname_is(FuncInternal1～4)")と宣言している。ここで、"funcname_is(FuncInternal1～4)")と定義することにより、この構造体のメンバに渡された値をFuncInternal1～4関数(２２６)のアドレスとして認識し、ログ取得処理のアドレスに置換する。また、ログ取得処理時に本来の非エクスポート関数を呼ぶためにオリジナルの値を保存する。これにより、エクスポートされていないモジュール内部の関数群のログを取得可能となる。

【００４４】

図１５は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図３とは、隠し関数のログ取得用のコードが追加されている点で異なっている。A.DLLにある、FuncAD(２３１)は、エクスポートされていない関数であり、そのため、Import Address Table(２３０)にも、存在しない。この時funcname_isをFuncADに対して定義すると、C.DLLにログ取得のためのFuncCAD(２３２)が生成され、本来FuncADのアドレスが返されるfuncname_isで定義されたパラメータの値を置換する場合に、FuncCADのアドレスが使用される。

【００４５】

図１６は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１４の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【００４６】

処理が開始されると(ステップＳ２１)、ログ取得を開始し、モジュール名、インターフェース名、関数／メソッド名をＨＤＤに保存する(ステップＳ２２)。次にログ取得コードは、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する(ステップＳ２３)。そして、関数定義にfuncname_isの設定が存在するか判別し(ステップＳ２４)、存在する場合には、funcname_isによって定義されている関数定義を関数定義ファイルから取得し、その定義に基づいてログ取得処理用のコードを生成する(ステップＳ２５)。そして、funcname_isが定義されている値を保存し、生成したログ取得コードのアドレスに置換する(ステップＳ２６)。

【００４７】

続いて、元の関数を呼び出す(ステップＳ２７)。関数からリターンすると、ログ取得コードはリターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する(ステップＳ２８)。そして、関数定義にfuncname_isの設定が存在するか判別し(ステップＳ２９)、存在する場合には、funcname_isによって定義されている関数の定義を関数定義ファイルから取得し、その定義に基づいてログ取得処理用のコードを生成する(ステップＳ３０)。そして、funcname_isが定義されている値を保存し、生成したログ取得コードのアドレスに置換する(ステップＳ３１)。この処理はユーザから終了の命令があると(ステップＳ３２)終了される。

【００４８】

funcname_isで設定されているパラメータが生成されたログ取得コードのアドレスに置換されると、その後そのアドレスを使用して呼び出される元の関数は、通常のログと同様に処理されるようになる（つまり、生成されたログ取得コードは、エクスポートされていない関数を呼び出し、処理を実行させ、受け取った実行結果を渡すとともに、エクスポートされていない関数を呼び出す際の所定の情

報と、実行結果を受け取った際の所定の情報とをログとして記録する）。

【００４９】

図１７は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１４の定義を行わない場合に取得されたログデータの図である。関数定義ファイルに於いて設定を行わない場合は、コールバック関数、エクスポートされていない内部関数のログは取得できない。

【００５０】

図１８は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１４の定義より取得されたログデータの図である。関数定義ファイルに於いて、コールバック関数、エクスポートされていない内部関数を取得するように設定しているので、FuncCallBack関数や、FuncInternal関数、FuncInternal4関数等の非エクスポート関数のログが取得可能となる。

【００５１】

このように本実施形態によれば、通常の方法では取得できない非エクスポート関数のログを取得可能になるという効果が得られる。

【００５２】

【第４の実施形態】

本実施形態では、通常の方法では取得できない関数のログとして、可変長配列のパラメータのログを取得する場合について述べる。

【００５３】

図１９は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて可変長配列のパラメータをログとして取得するためのＩＤＬによる記述である。

【００５４】

FuncArrayIs関数の定義に於いて、int* lpnParamに対してcustum(PAT_PARAM_ATTR_ID, "array_is(dwCount)")と宣言している（２４０）。ここで、" array_is(dwCount)"と定義することにより、このポインタ・パラメータは第一パラメータであるdwCount個分のint型配列であると扱われる。そして、このポインタ・パラメータが指すポインタをint型のdwCount個の配列とし、データをログに保存する。

【００５５】

図２０は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１９の様に関数が定義されている時のログを取得する場合の流れ図である。

【００５６】

処理が開始されると(ステップＳ４１)、ログ取得を開始し、モジュール名、インターフェース名、関数／メソッド名をＨＤＤに保存する(ステップＳ４２)。次に関数定義に可変長配列取得(array_is)の設定が存在するか判別し(ステップＳ４３)、存在する場合には、ポインタ・パラメータで定義されているパラメータを配列の定義として扱い(ステップＳ４４)、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する(ステップＳ４５)。そして、続いて、元の関数を呼び出す(ステップＳ４６)。関数からリターンすると、ログ取得コードは関数定義に可変長配列取得(array_is)の設定が存在するか判別し(Ｓ４７)、存在する場合には、ポインタ・パラメータで定義されているパラメータを配列の定義として扱い(ステップＳ４８)、リターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する(ステップＳ４９)。この処理はユーザから終了の命令があると(ステップＳ５０)終了される。

【００５７】

図２１は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１９の定義を行わない場合に取得されたログデータ(２５０)と、図１９の定義により取得されたログデータ(２５１)の図である。関数定義ファイルに於いて設定を行わない場合は、ポインタ・パラメータは配列としては扱えないので、先頭のデータのみが取得されているが、関数定義ファイルに於いて設定が行われている場合では、ポインタ・パラメータは配列として扱われるため、配列の全データが取得されている。

【００５８】

このように本実施形態によれば、通常の方法では取得できない関数のログとして、可変長配列のパラメータのログが取得可能になるという効果が得られる。

【００５９】

【第５の実施形態】

本実施形態では、通常の関数定義では、パラメータを取得できない関数について、ログを取得する場合について説明する。

【００６０】

図２２は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、通常の関数定義では、パラメータを取得できない関数の例である。

【００６１】

構造体として、STRUCTSIZE1，STRUCTSIZE2，STRUCTSIZE3の３種類が定義されており、各構造体のDWORD dwSizeメンバには、自分自身の構造体のサイズを入れる必要がある。FuncGetData関数は、第一パラメータdwKindと、第二パラメータlpBufに上記３種類のどの構造体のポインタが渡されているかを設定する。FuncGetData関数は、第一パラメータが１の場合には、lpBufをSTRUCTSIZE1のポインタとして扱って処理を行い、第一パラメータが２の場合には、lpBufをSTRUCTSIZE2のポインタとして扱って処理を行い、第一パラメータが３の場合には、lpBufをSTRUCTSIZE3のポインタとして扱って処理を行う。この場合、通常の関数定義により、FuncGetDataを定義すると、lpBufはvoid型のポインタとなり、データを取得することはできない。

【００６２】

図２３は、図２２に於いて、STRUCTSIZE1，STRUCTSIZE2，STRUCTSIZE3の各構造体がどのようにメモリを使用するかのメモリ図である。構造体STRUCTSIZE1(２６０)の、各メンバは、オフセット０ｘ００００にDWORD dwSize(２６１)、０ｘ０００４にDWORD dwParam1(２６２)、０ｘ０００８にDWORD dwParam2、０ｘ０００ＣにDWORD dwParam3が存在する。構造体STRUCTSIZE2(２６５)はDWORD dwSize～DWORD dwParam3(２６６～２６９)までは、STRUCTSIZE1と同様であり、０ｘ００１０にDWORD dwParam4(２７０)が存在する。STRUCTSIZE3(２７１)はDWORD dwSize～DWORD dwParam4(２７２～２７６)までは、STRUCTSIZE2と同様であり、０ｘ００１４にDWORD dwParam5(２７７)が存在する。このように、STRUCTSIZE3のdwSize～dwParam4までのメモリ配置は、STRUCTSIZE2と同様であり、STRUCTSIZE2のdwSize～dwP

aram3までのメモリ配置は、STRUCTSIZE1と同様になっている。

【００６３】

図２４は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図２２の様な関数のパラメータのログを取得するためのＩＤＬによる記述である。

【００６４】

本来の関数では、STRUCTSIZE1, STRUCTSIZE2, STRUCTSIZE3と分かれていた構造体をSTRUCTSIZEという１個の構造体として定義する(２９１)。この定義はSTRUCTSIZE3と同様であり、これは、図２３でのメモリ配置から、STRUCTSIZE3は、STRUCTSIZE1、STRUCTSIZE2と同様のメモリ配置部分を持つからである。そして、この構造体のメンバであり、構造体のサイズを表す、DWORD dwSizeに対して、[custum(PAT_PARAM_ID, "structsize_is()")]を設定する。この事で、ログ取得処理は構造体のデータ解析を行う場合に、構造体のサイズを知ることが可能となる。また、FuncGetData関数において、第二パラメータのlpBufをSTRUCTSIZE型として定義しておく。こうすることで、このパラメータをログとして保存する場合には、STRUCTSIZE型として処理が行われる。仮に、本手法を用いず、STRUCTSIZE3のデータを取得するように関数定義ファイルに記述した場合には、GetFuncDataがdwKind=１や２で呼ばれた場合には、STRUCTSIZE1やSTRUCTSIZE2のデータが存在することになり、dwParam3やdwParam4のデータをログに取得しようとしてしまうために、メモリ例外が発生する。

【００６５】

図２５は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【００６６】

処理が開始されると(ステップＳ６１)、ログ取得を開始し、モジュール名、インターフェース名、関数／メソッド名をＨＤＤに保存する(ステップＳ６２)。次に関数定義に構造体のサイズ指定(structsize_is)の設定が存在するか判別し(ステップＳ６３)、存在する場合には、設定されている構造体パラメータのデータ

をサイズ指定(structsize_is)の設定で定義されているデータサイズ範囲内で解析する(ステップＳ６４)。そして、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する(ステップＳ６５)。そして、続いて、元の関数を呼び出す(ステップＳ６６)。関数からリターンすると、関数定義に構造体のサイズ指定(structsize_is)の設定が存在するか判別し(ステップＳ６７)、存在する場合には、設定されている構造体パラメータのデータをサイズ指定(structsize_is)の設定で定義されているデータサイズ範囲内で解析する(ステップＳ６８)。そして、リターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する(ステップＳ６９)。この処理はユーザから終了の命令があると(ステップＳ７０)終了される。

【００６７】

図２６は、図２５で行われる構造体パラメータ解析の詳細をメモリ配置を基に表した図である。

【００６８】

実際のプログラムでGetFuncDataがdwKind=1で使用された場合には、STRUCTSIZE1構造体(３００)が使用され、その時の各メンバ(３０１～３０４)のメモリ使用は、図のようになっている。図２４の様に関数定義がなされている場合には、本ソフトウェア評価システムでは、STRUCTSIZE構造体(３０５)として認識され、その時の各メンバ(３０６～３１１)のメモリ使用は図のようになると推測する。そして、サイズ指定(structsize_is)の設定が存在する場合には、dwSizeの値を調べることで、構造体のサイズがdwSizeバイトであると知ることが可能となり、STRUCTSIZE構造体のうち、dwSizeバイト分(３１２)のデータのみをログとして取得する。

【００６９】

図２７は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図２４の定義により取得されたログデータの図である。通常の関数定義では、void*となり、ポインタのみしか取得できない構造体のデータが、使用される構造体の種類により、その構造体のデータがログとして取得されている。

【００７０】

このように、本実施形態によれば、通常の方法では取得できないパラメータのログを取得可能になるという効果が得られる。

【００７１】

【第６の実施形態】

図２８は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、通常の関数定義では、パラメータを取得できない関数の例である。

【００７２】

構造体として、STRUCTKIND1, STRUCTKIND2, STRUCTKIND3の３種類が定義されている。GetFuncData関数は、第一パラメータdwKindに第二パラメータlpBufに上記３種類のどの構造体のポインタが渡されているかを設定する。FuncGetData関数は、第一パラメータが１の場合には、lpBufをSTRUCTKIND1のポインタとして扱って処理を行い、第一パラメータが２の場合には、lpBufをSTRUCTKIND2のポインタとして扱って処理を行い、第一パラメータが３の場合には、lpBufをSTRUCTKIND3のポインタとして扱って処理を行う。この場合、通常の関数定義により、FuncGetDataを定義すると、lpBufはvoid型のポインタとなり、データを取得することはできない。

【００７３】

図２９は、図２８に於いて、STRUCTKIND1, STRUCTKIND2, STRUCTKIND3の各構造体がどのようなメモリ配置になるかを示した図である。構造体STRUCTKIND1(３３０)の、各メンバは、オフセット０ｘ０００００にchar chParam(３３１)、０ｘ０００１にDWORD dwParam(３３２)、０ｘ０００５にshort shParam(３３３)が存在する。構造体STRUCTKIND2(３３４)の、各メンバは、オフセット０ｘ０００００にshort shParam(３３５)、０ｘ０００２にDWORD dwParam(３３６)、０ｘ０００６にchar chParam(３３７)が存在する。構造体STRUCTKIND3(３３８)の、各メンバは、オフセット０ｘ０００００にchar chParam(３３９)、０ｘ０００１にshort shParam(３４０)、０ｘ０００３にDWORD dwParam(３４１)、０ｘ０００７にlong lParam(３４２)、０ｘ０００Ｂにint nParamが存在する。このように、各構造体にサイズの情報が無く、構造体データのメモリ構造も異なるため、第５の実施形態で示し

た方法は使用できない。

【００７４】

図３０は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図２８の様な関数のパラメータのログを取得するためのＩＤＬによる記述である。

【００７５】

各構造体は、通常の方法で定義されている。FuncGetData関数の第二パラメータvoid* lpBufに対して[custum(PAT_PARAM_ID, "structkind_is(dwKind:1:STRUCTKIND1*, 2:STRUCTKIND2*, 3:STRUCTKIND3*)")]と設定している。これによりlpBufのデータ型は、第一パラメータdwKindの値が１の場合には、STRUCTKIND1*として、２の場合には、STRUCTKIND2*として、３の場合には、STRUCTKIND3*として扱われ、ログとして保存される。

【００７６】

図３１は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図３０の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【００７７】

処理が開始されると(ステップＳ８１)、ログ取得を開始し、モジュール名、インターフェース名、関数／メソッド名をメモリに保存する(ステップＳ８２)。次に関数定義に構造体の種類指定(structkind_is)の設定が存在するか判別し(ステップＳ８３)、存在する場合には、設定されている構造体パラメータのデータを種類指定(structkind_is)の設定で定義されているデータ型として解析する(ステップＳ８４)。そして、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をメモリに保存する(ステップＳ８５)。そして、続いて、元の関数を呼び出す(ステップＳ８６)。関数からリターンすると、関数定義に構造体の種類指定(structkind_is)の設定が存在するか判別し(ステップＳ８７)、存在する場合には、設定されている構造体パラメータのデータを種類指定(structkind_is)の設定で定義されているデータ型として解析する(ステップＳ８８)。そして、リターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内

容をメモリに保存する(ステップＳ８９)。この処理はユーザから終了の命令があると(ステップＳ９０)終了される。

【００７８】

図３２は、本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図３０の定義により取得されたログデータの図である。通常の関数定義では、void*となり、ポインタのみしか取得できない構造体のデータが、使用される構造体の種類により、その構造体のデータがログとして取得されている。

【００７９】

このように本実施形態によれば、通常の方法では取得できないパラメータのログを取得可能になるという効果が得られる。

【００８０】

【他の実施形態】

なお、本発明は、複数の機器（例えばホストコンピュータ，インタフェイス機器、リーダ、プリンタなど）から構成されるシステムに適用しても、一つの機器からなる装置（例えば、複写機、ファクシミリ装置など）に適用してもよい。

【００８１】

また、本発明の目的は、前述した実施形態の機能を実現するソフトウェアのプログラムコードを記録した記憶媒体を、システムあるいは装置に供給し、そのシステムあるいは装置のコンピュータ（またはＣＰＵやＭＰＵ）が記憶媒体に格納されたプログラムコードを読出し実行することによっても、達成されることは言うまでもない。

【００８２】

この場合、記憶媒体から読出されたプログラムコード自体が前述した実施形態の機能を実現することになり、そのプログラムコードを記憶した記憶媒体は本発明を構成することになる。

【００８３】

プログラムコードを供給するための記憶媒体としては、例えば、フロッピ（登録商標）ディスク、ハードディスク、光ディスク、光磁気ディスク、ＣＤ－ＲＯ

Ｍ、ＣＤ－Ｒ、磁気テープ、不揮発性のメモリカード、ＲＯＭなどを用いることができる。

【００８４】

また、コンピュータが読出したプログラムコードを実行することにより、前述した実施形態の機能が実現されるだけでなく、そのプログラムコードの指示に基づき、コンピュータ上で稼働しているＯＳ（オペレーティングシステム）などが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【００８５】

さらに、記憶媒体から読出されたプログラムコードが、コンピュータに挿入された機能拡張ボードやコンピュータに接続された機能拡張ユニットに備わるメモリに書込まれた後、そのプログラムコードの指示に基づき、その機能拡張ボードや機能拡張ユニットに備わるＣＰＵなどが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【００８６】

【発明の効果】

以上説明したように本発明によれば、複数にモジュール分けされたソフトウェアの処理ログを容易に取得でき、かつ、ソフトウェアの障害の原因の解析のための工数を削減することが可能となる。

【図面の簡単な説明】

【図１】

本発明の第１の実施形態にかかるログ取得方法を実現するコンピュータ（ソフトウェア評価システム）の構成をあらわす図である。

【図２】

本発明の第１の実施形態にかかるログ取得方法を説明するためにあらわした、関数ロード時の通常のメモリ構成をあらわす図である。

【図３】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価シ

ステムのＩＡＴ　Ｐａｔｃｈ使用時のメモリ構成をあらわす図である。

【図４Ａ】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのＩＡＴ　Ｐａｔｃｈを使用時の状態を示す図である。

【図４Ｂ】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのログ取得処理の流れを示す図である。

【図５】

本発明の第１の実施形態にかかるソフトウェア評価システムのＩＡＴ　Ｐａｔｃｈを使用時の内部構成をあらわす図である。

【図６】

本発明の第１の実施形態にかかるログ取得方法を説明するためにあらわした、ＣＯＭサーバのインターフェースのインスタンス作成時の通常のメモリ構成をあらわす図である。

【図７】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのＶＴａｂｌｅ　Ｐａｔｃｈ使用時のメモリ構成をあらわす図である。

【図８Ａ】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのＶＴａｂｌｅ　Ｐａｔｃｈを使用時の状態を示す図である。

【図８Ｂ】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのログ取得処理の流れを示す図である。

【図９】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの、内部構成をあらわす図である。

【図１０】

本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義の一例を示す図である。

【図１１】

本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて、ポインタ・パラメータに対してバイナリ取得を指定することで、ポインタ・パラメータのデータの実体をログとして取得する為のＩＤＬによる記述を示す図である。

【図１２】

本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１１の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図１３】

本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１１の定義により取得されたログデータの図である。

【図１４】

本発明の第３の本実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて、コールバック関数等のエクスポートされていない関数をログとして取得するためのＩＤＬによる記述を示す図である。

【図１５】

本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図である。

【図１６】

本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１４の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図１７】

本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１４の定義を行わない場合に取得されたログデータの図である。

【図１８】

本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価シ

ステムにおいて、図１４の定義より取得されたログデータの図である。

【図１９】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義に於いて、可変長配列のパラメータをログとして取得するためのＩＤＬによる記述を示す図である。

【図２０】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１９の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図２１】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１９の定義を行わない場合に取得されたログデータ（２５０）と、図１９の定義により取得されたログデータ（２５１）の図である。

【図２２】

本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、通常の関数定義では、パラメータを取得できない関数の例を示す図である。

【図２３】

図２２に於いて、STRUCTSIZE1，STRUCTSIZE2，STRUCTSIZE3の各構造体がどのようにメモリを使用するかを示す図である。

【図２４】

本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図２２の様な関数のパラメータのログを取得するためのＩＤＬによる記述を示す図である。

【図２５】

本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図２６】

図２５で行われる構造体パラメータ解析の詳細をメモリ配置をもとに表した図である。

【図２７】

本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図２４の定義により取得されたログデータの図である。

【図２８】

本発明の第６の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、通常の関数定義では、パラメータを取得できない関数の例を示す図である。

【図２９】

図２８に於いて、STRUCTKIND1，STRUCTKIND2，STRUCTKIND3の各構造体がどのようなメモリ配置になるかを示した図である。

【図３０】

本発明の第６の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図２８の様な関数のパラメータのログを取得するためのＩＤＬによる記述を示す図である。

【図３１】

本発明の第６の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで、図３０の様に関数が定義されている時のログを取得する場合の流れを示す図である。

【図３２】

本発明の第６の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図３０の定義により取得されたログデータの図である。

【符号の説明】

１：ＣＰＵ

２：チップセット

３：ＲＡＭ

４：ハードディスクコントローラ

５：ディスプレイコントローラ

６：ハードディスクドライブ

７：ＣＤ－ＲＯＭドライブ

８：ディスプレイ

１１：ＣＰＵとチップセットを繋ぐ信号線

１２：チップセットとＲＡＭを繋ぐ信号線

１３：チップセットと各種周辺機器とを繋ぐ周辺機器バス

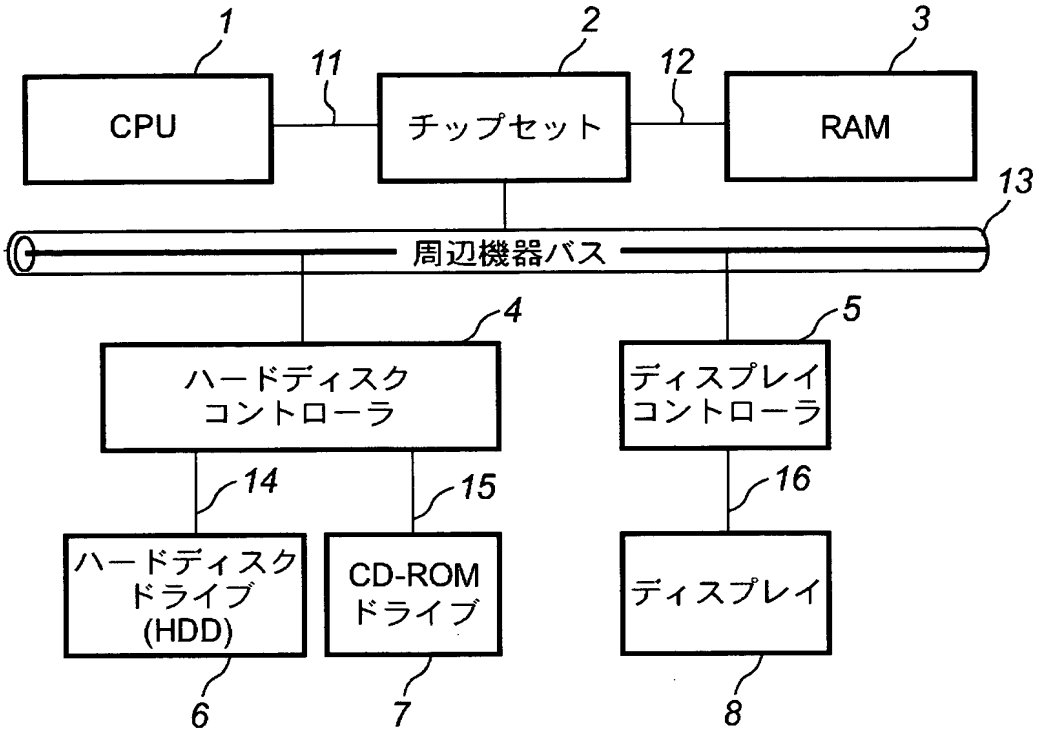
１４：ハードディスクコントローラとハードディスクドライブを繋ぐ信号線

１５：ハードディスクコントローラとＣＤ－ＲＯＭドライブを繋ぐ信号線

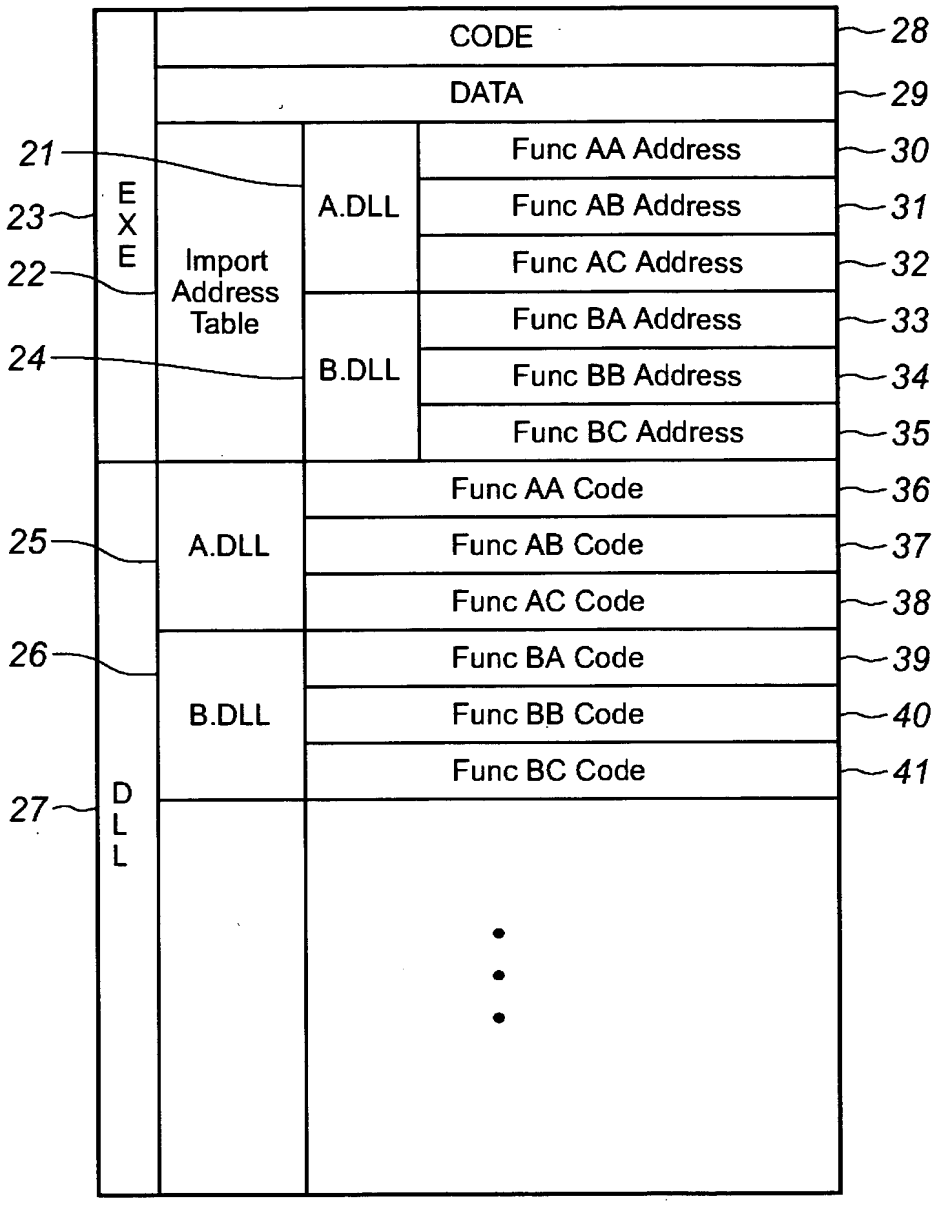
１６：ディスプレイコントローラとディスプレイを繋ぐ信号線

【書類名】　図面

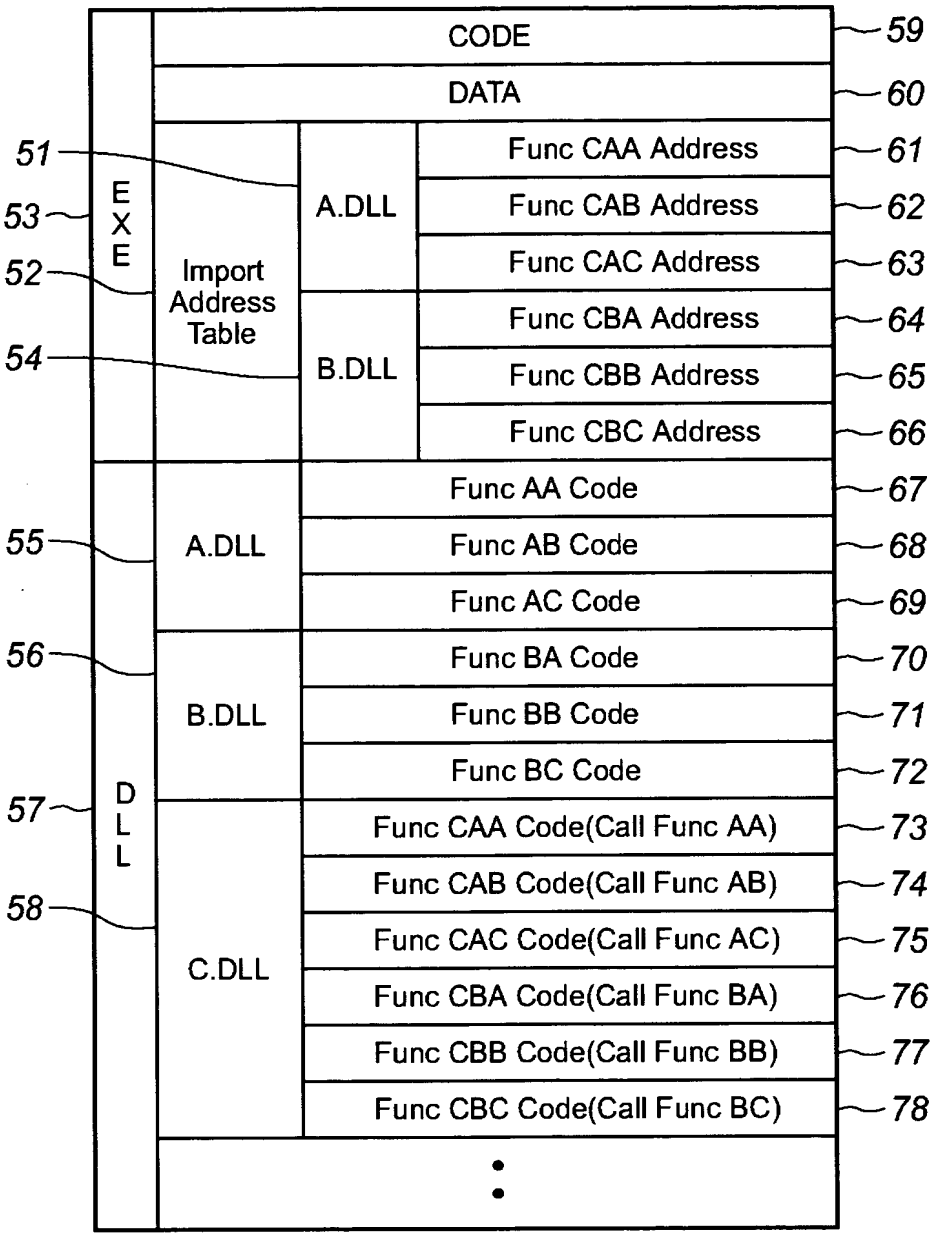
【図１】


1
2
3
11
12
CPU
チップセット
RAM
13
周辺機器バス
4
5
ハードディスク
コントローラ
ディスプレイ
コントローラ
14
15
16
ハードディスク
ドライブ
(HDD)
CD-ROM
ドライブ
ディスプレイ
6
7
8

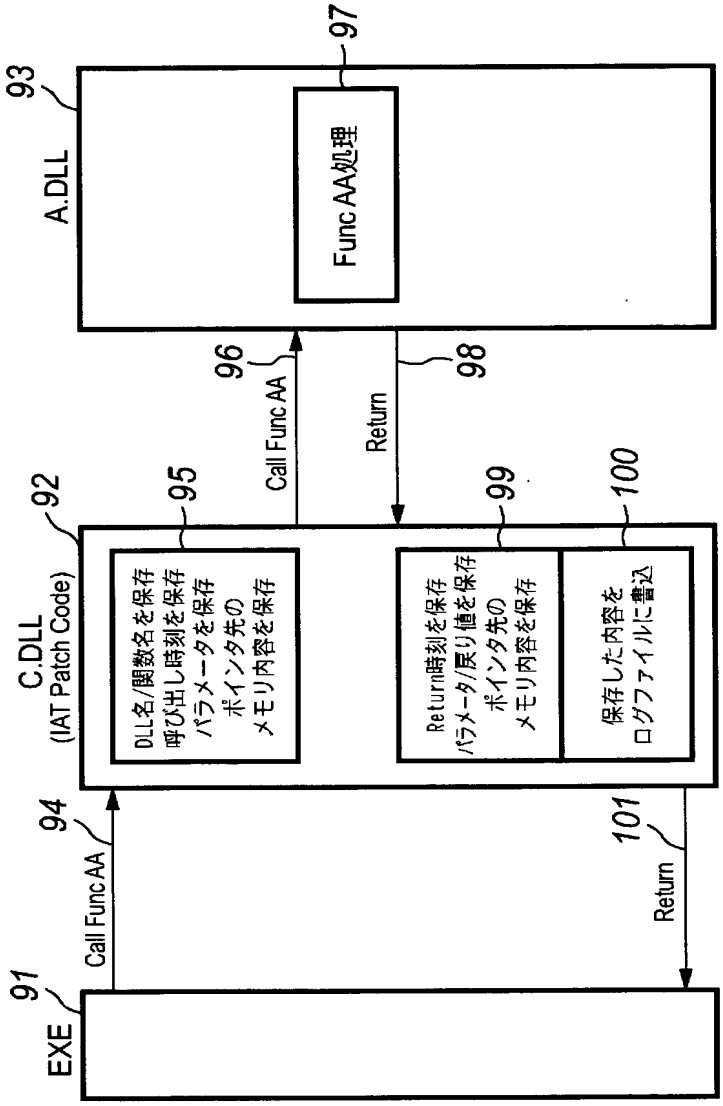
【図２】


CODE
28
DATA
29
21
23
22
24
EXE
Import Address Table
A.DLL
B.DLL
Func AA Address
30
Func AB Address
31
Func AC Address
32
Func BA Address
33
Func BB Address
34
Func BC Address
35
25
26
27
DLL
A.DLL
B.DLL
Func AA Code
36
Func AB Code
37
Func AC Code
38
Func BA Code
39
Func BB Code
40
Func BC Code
41

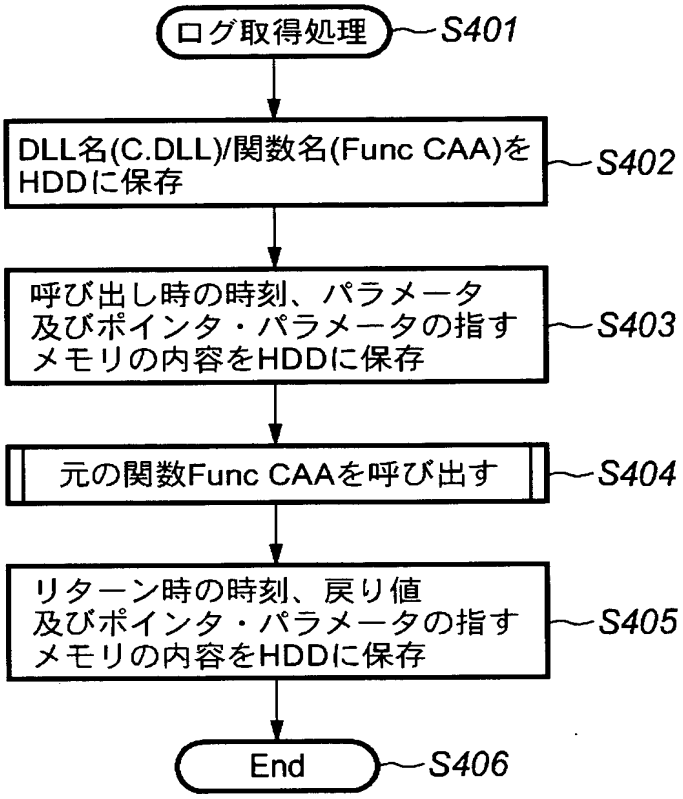
【図３】


CODE
DATA
EXE
Import Address Table
A.DLL
Func CAA Address
Func CAB Address
Func CAC Address
B.DLL
Func CBA Address
Func CBB Address
Func CBC Address
DLL
A.DLL
Func AA Code
Func AB Code
Func AC Code
B.DLL
Func BA Code
Func BB Code
Func BC Code
C.DLL
Func CAA Code(Call Func AA)
Func CAB Code(Call Func AB)
Func CAC Code(Call Func AC)
Func CBA Code(Call Func BA)
Func CBB Code(Call Func BB)
Func CBC Code(Call Func BC)
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78

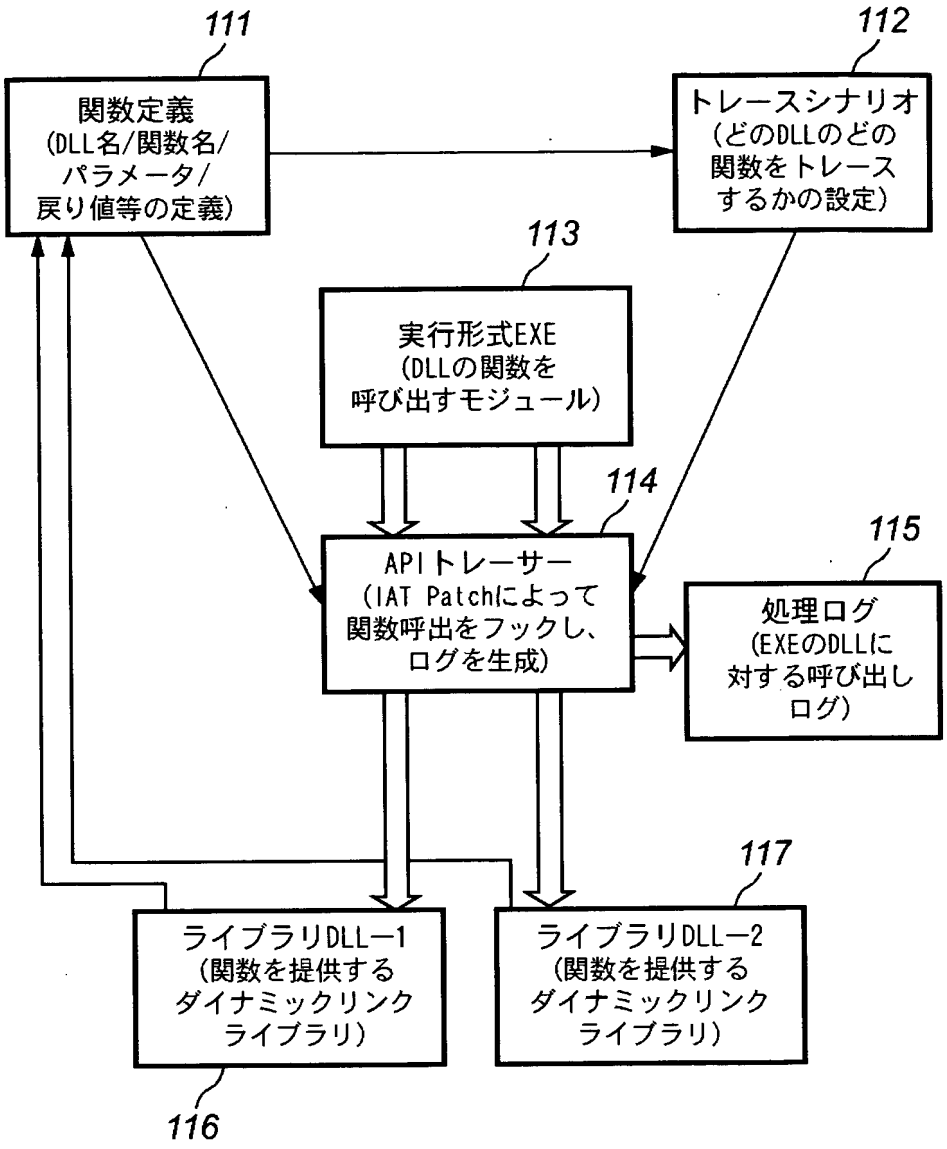
【図４Ａ】


97
93
A.DLL
Func AA処理
96
Call Func AA
Return
98
92
95
99
100
C.DLL
(IAT Patch Code)
DLL名/関数名を保存
呼び出し時刻を保存
パラメータを保存
ポインタ先の
メモリ内容を保存
Return時刻を保存
パラメータ/戻り値を保存
ポインタ先の
メモリ内容を保存
保存した内容を
ログファイルに書込
94
Call Func AA
101
Return
91
EXE

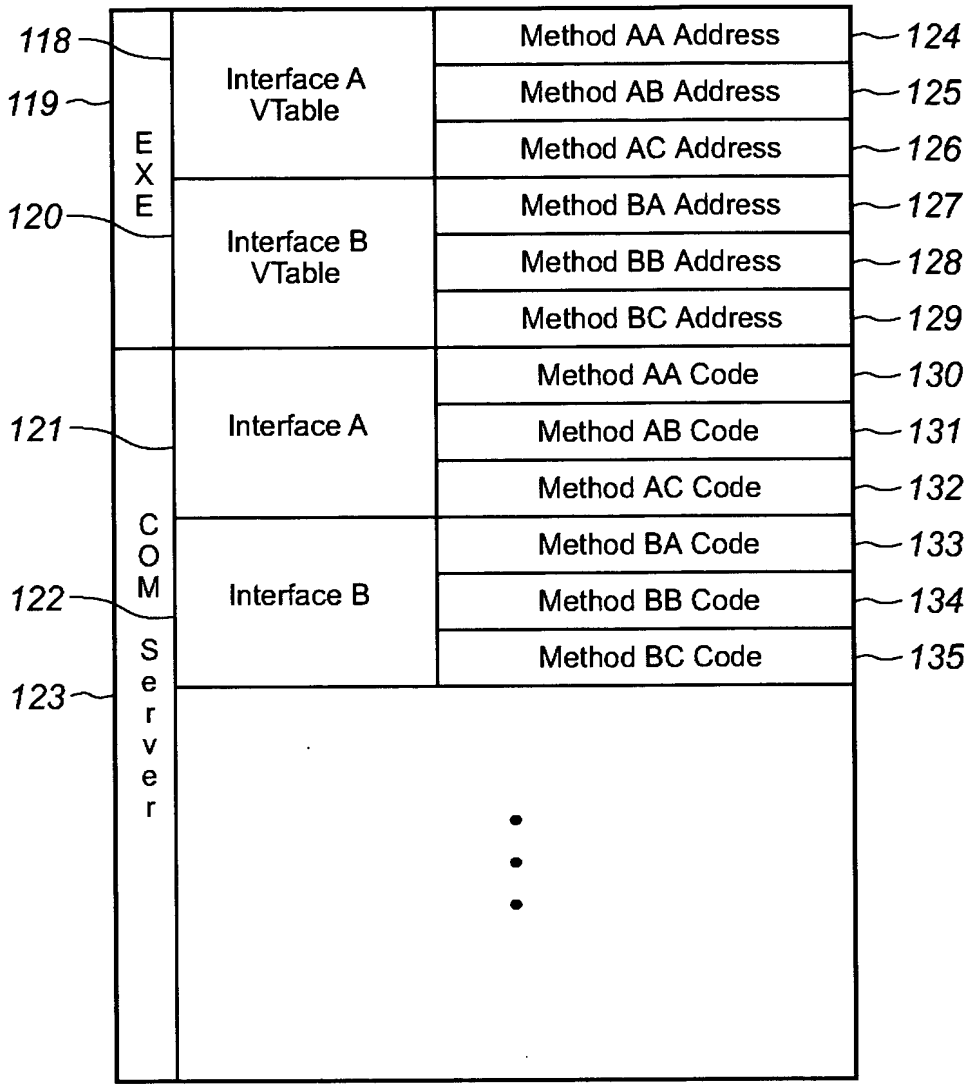
【図４Ｂ】


ログ取得処理
S401
DLL名(C.DLL)/関数名(Func CAA)を
HDDに保存
S402
呼び出し時の時刻、パラメータ
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S403
元の関数Func CAAを呼び出す
S404
リターン時の時刻、戻り値
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S405
End
S406

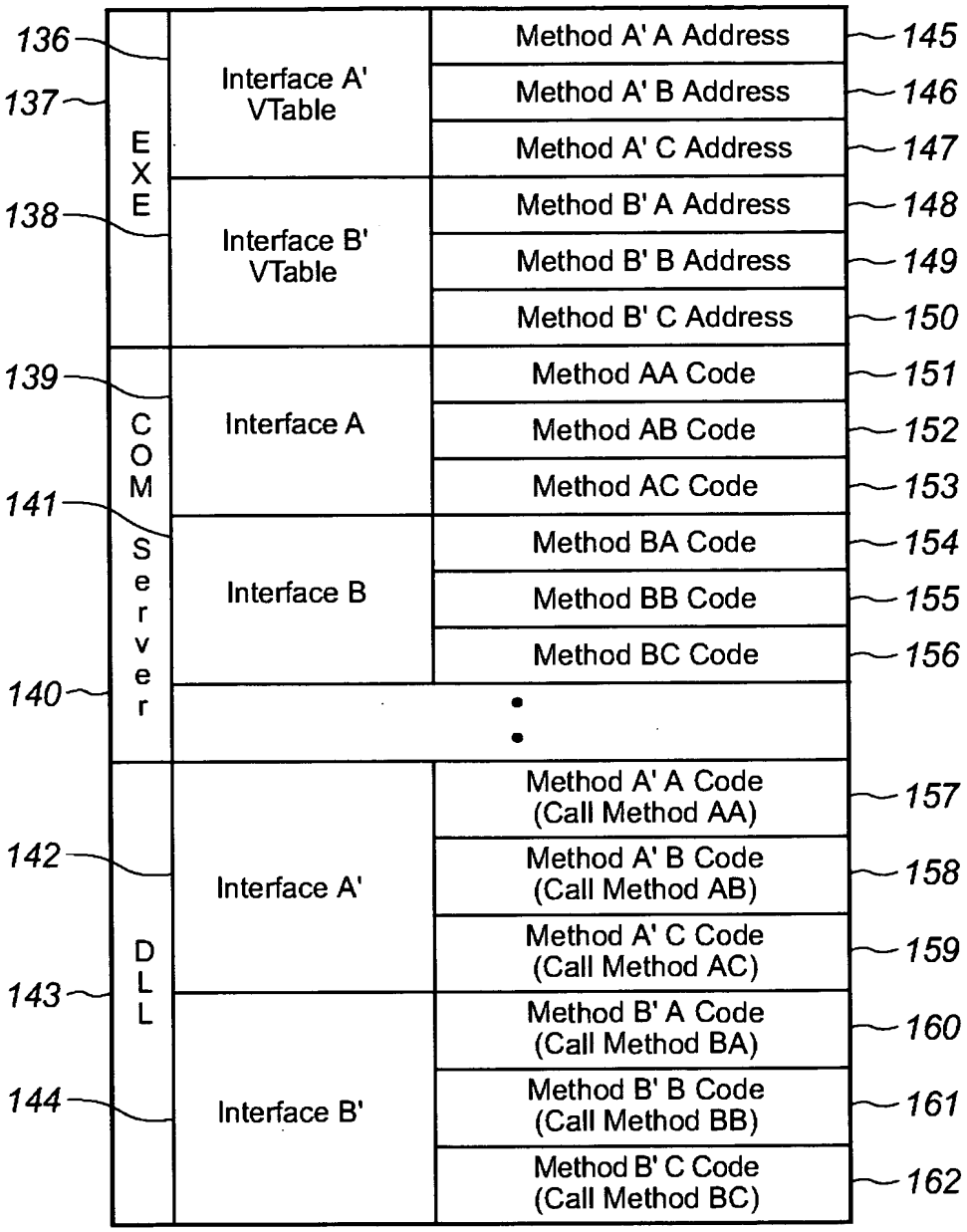
【図５】


111
関数定義
(DLL名/関数名/
パラメータ/
戻り値等の定義)
112
トレースシナリオ
(どのDLLのどの
関数をトレース
するかの設定)
113
実行形式EXE
(DLLの関数を
呼び出すモジュール)
114
115
APIトレーサー
(IAT Patchによって
関数呼出をフックし、
ログを生成)
処理ログ
(EXEのDLLに
対する呼び出し
ログ)
117
ライブラリDLL−1
(関数を提供する
ダイナミックリンク
ライブラリ)
ライブラリDLL−2
(関数を提供する
ダイナミックリンク
ライブラリ)
116

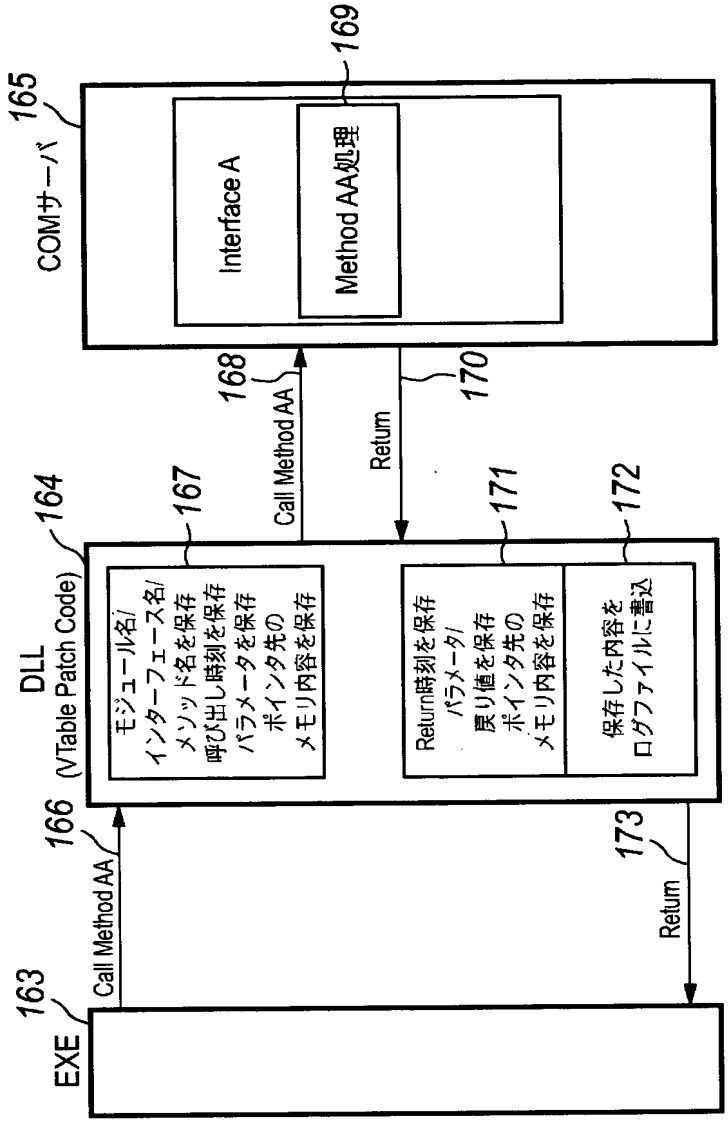
【図６】


118
119
120
121
122
123
EXE
COM Server
Interface A VTable
Interface B VTable
Interface A
Interface B
Method AA Address
Method AB Address
Method AC Address
Method BA Address
Method BB Address
Method BC Address
Method AA Code
Method AB Code
Method AC Code
Method BA Code
Method BB Code
Method BC Code
124
125
126
127
128
129
130
131
132
133
134
135

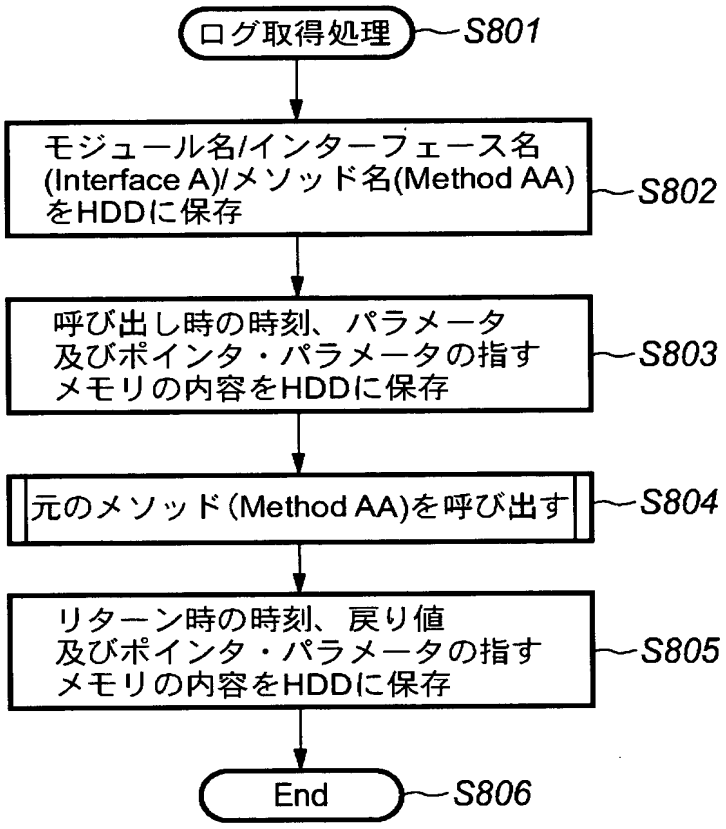
【図７】


136
137
138
139
141
140
142
143
144
EXE
COM Server
DLL
Interface A' VTable
Interface B' VTable
Interface A
Interface B
Interface A'
Interface B'
Method A' A Address
Method A' B Address
Method A' C Address
Method B' A Address
Method B' B Address
Method B' C Address
Method AA Code
Method AB Code
Method AC Code
Method BA Code
Method BB Code
Method BC Code
Method A' A Code (Call Method AA)
Method A' B Code (Call Method AB)
Method A' C Code (Call Method AC)
Method B' A Code (Call Method BA)
Method B' B Code (Call Method BB)
Method B' C Code (Call Method BC)
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162

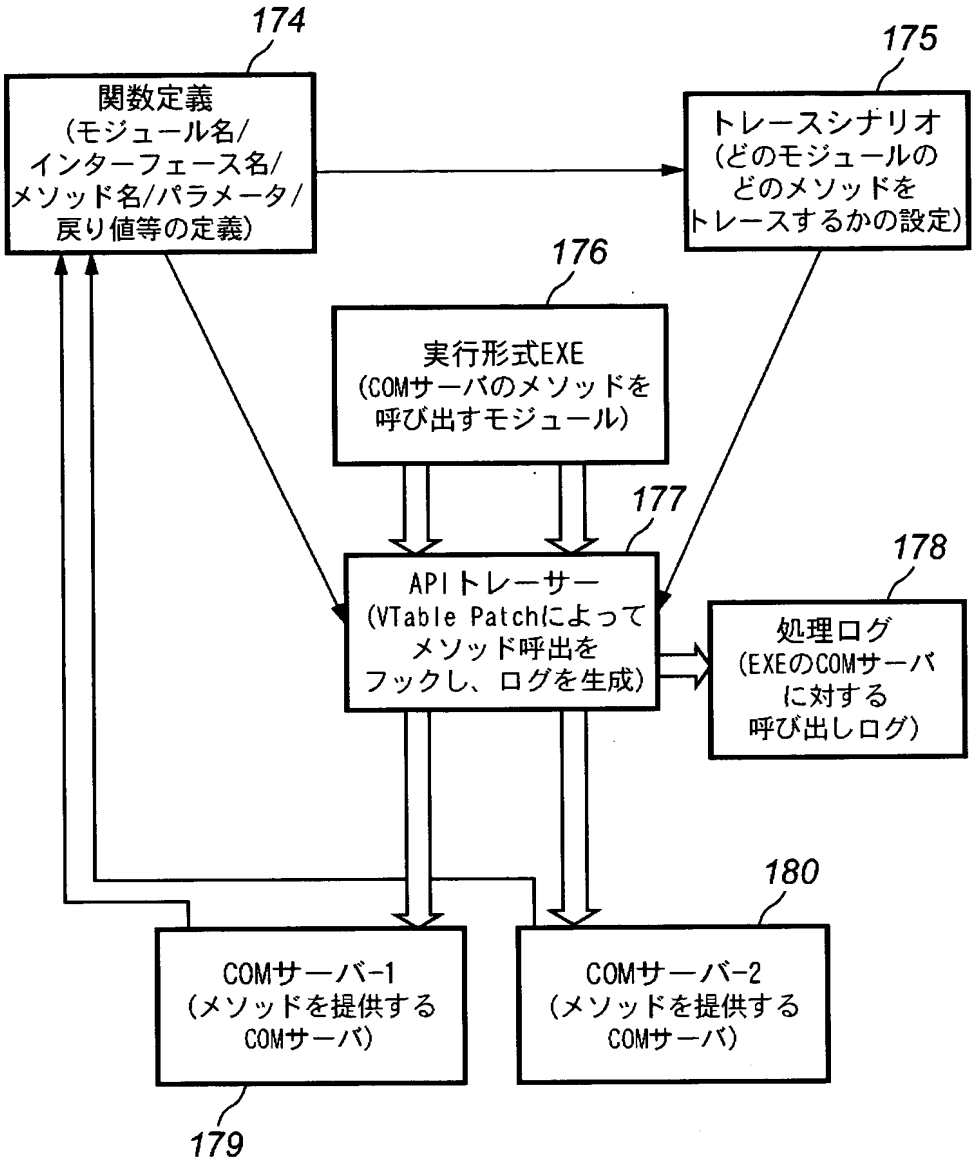
【図８Ａ】


165
COMサーバ
169
Interface A
Method AA処理
168
Call Method AA
170
Return
164
DLL
(VTable Patch Code)
167
モジュール名/
インターフェース名/
メソッド名を保存
呼び出し時刻を保存
パラメータを保存
ポインタ先の
メモリ内容を保存
171
Return時刻を保存
パラメータ/
戻り値を保存
ポインタ先の
メモリ内容を保存
172
保存した内容を
ログファイルに書込
166
Call Method AA
173
Return
163
EXE

【図８Ｂ】


ログ取得処理
S801
モジュール名/インターフェース名(Interface A)/メソッド名(Method AA)をHDDに保存
S802
呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をHDDに保存
S803
元のメソッド(Method AA)を呼び出す
S804
リターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をHDDに保存
S805
End
S806

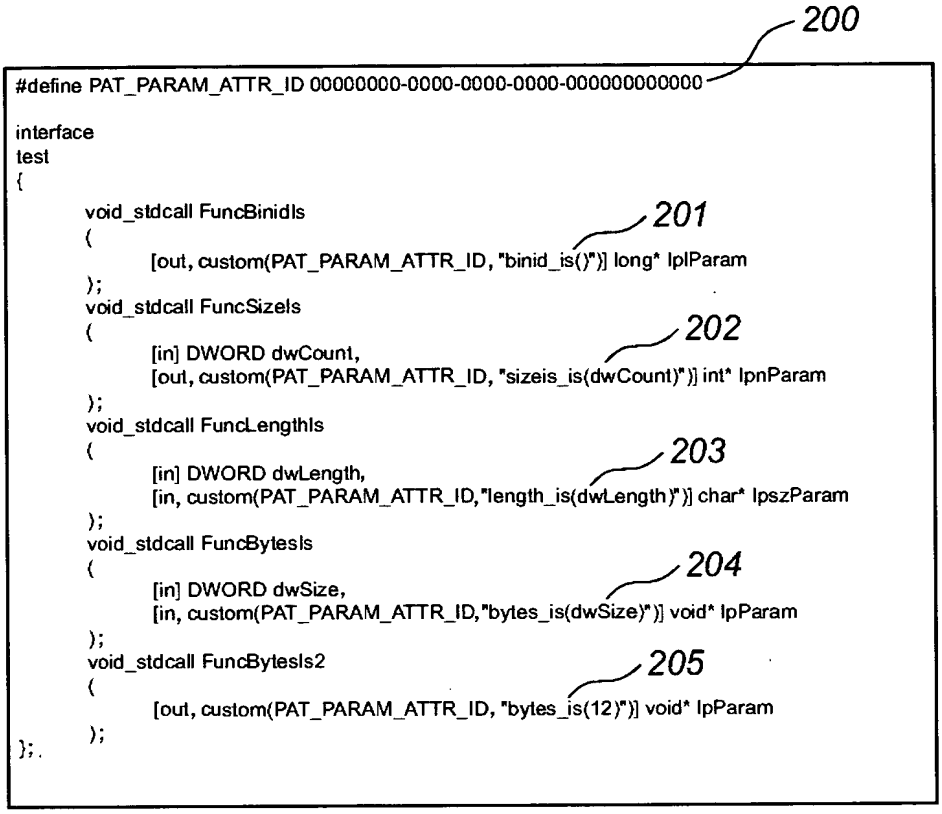
【図９】


174
関数定義
(モジュール名/
インターフェース名/
メソッド名/パラメータ/
戻り値等の定義)
175
トレースシナリオ
(どのモジュールの
どのメソッドを
トレースするかの設定)
176
実行形式EXE
(COMサーバのメソッドを
呼び出すモジュール)
177
APIトレーサー
(VTable Patchによって
メソッド呼出を
フックし、ログを生成)
178
処理ログ
(EXEのCOMサーバ
に対する
呼び出しログ)
180
COMサーバ-1
(メソッドを提供する
COMサーバ)
COMサーバ-2
(メソッドを提供する
COMサーバ)
179

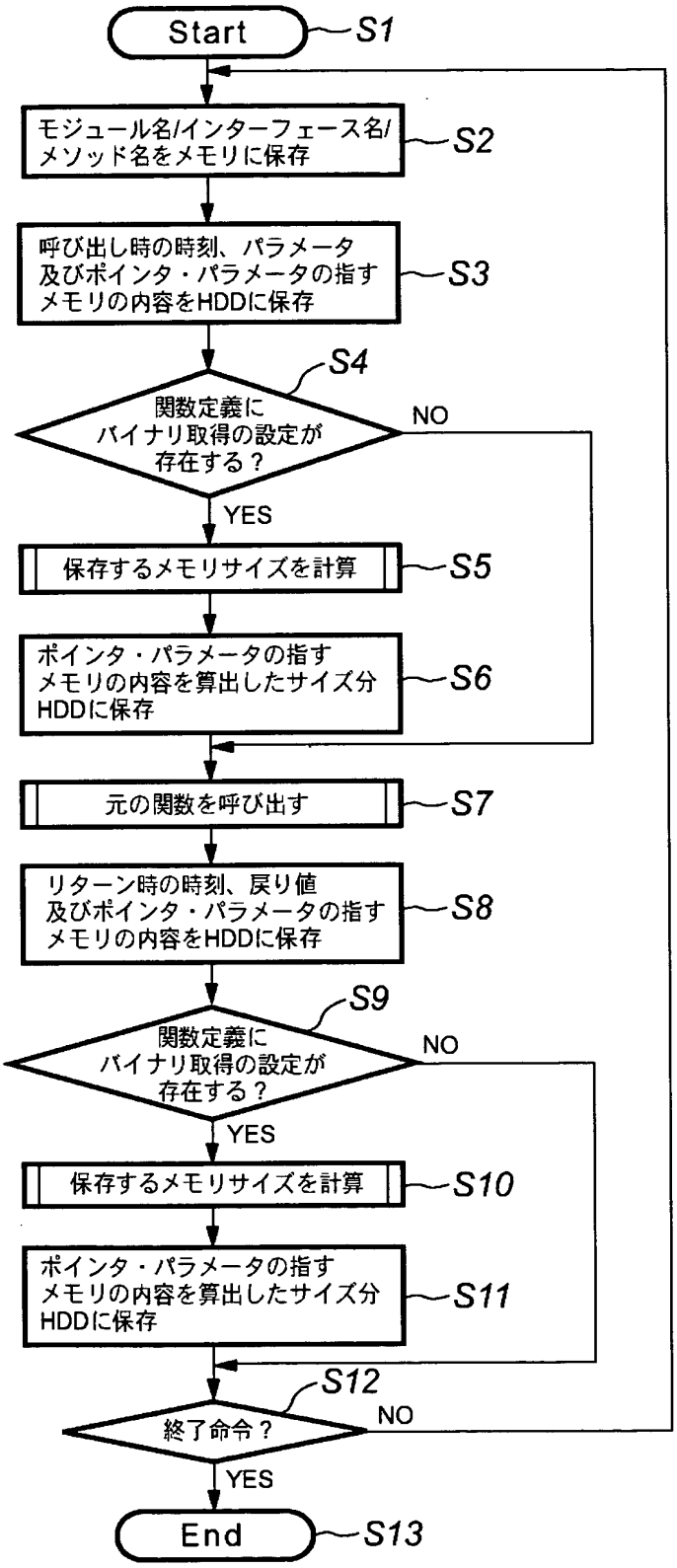
【図１０】

```
[
    uuid(58DB5633-0694-4340-97CE-4E1AC6BFFBA7),    // TestDllStd
    helpstring("TestDllStd Type Library For PAT"),
    version(1.0)
]
library TestDllStd

        typedef [public] struct
        {
                char chParam;
                unsigned char uchParam;
                short sParam;
                unsigned short usParam;
                int nParam;
                unsigned int unParam;
                long lParam;
                unsigned long ulParam;
                double dbParam;
                float fParam;
        } TESTSTRUCT;
        typedef [public] TESTSTRUCT *LPTESTSTRUCT;
//DEFINE_GUID(GUID_PROGID, 0x8e037d65, 0xefa0,0x40e7, 0x91, 0x43, 0xef, 0x70,0x56, 0x94, 0x5b,
0x79);
 [
    uuid(8E037D65-EFA0-40e7-9143-EF7056945B79),
 ]  helpstring("TestDllStd.dll for PAT object."),

        interface
        test
        {
                char _stdcall FuncCharStd([in] char chParam);
                char* _stdcall FuncPCharStd([in, out] char* lpchParam);

                TESTSTRUCT _stdcall FuncStructStd[in]TESTSTRUCT TestStruct);
        };      LPTESTSTRUCT _stdcall FuncPStructStd([in, out]LPTESTSTRUCT lp TestStruct);
}
```

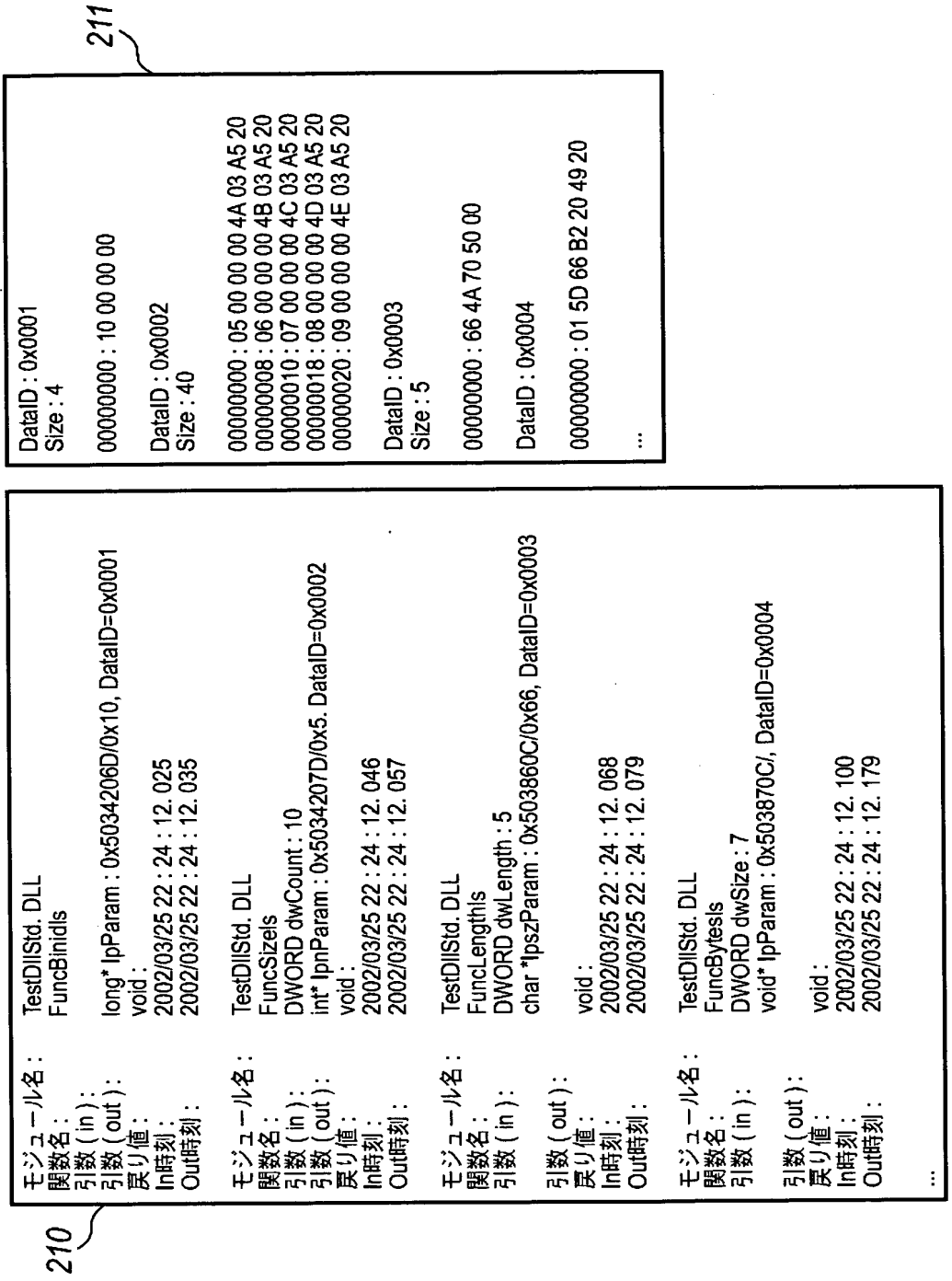
【図１１】


200
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000
interface
test
{
void_stdcall FuncBinidIs
(
[out, custom(PAT_PARAM_ATTR_ID, "binid_is()")] long* lplParam
);
201
void_stdcall FuncSizeIs
(
[in] DWORD dwCount,
[out, custom(PAT_PARAM_ATTR_ID, "sizeis_is(dwCount)")] int* lpnParam
);
202
void_stdcall FuncLengthIs
(
[in] DWORD dwLength,
[in, custom(PAT_PARAM_ATTR_ID,"length_is(dwLength)")] char* lpszParam
);
203
void_stdcall FuncBytesIs
(
[in] DWORD dwSize,
[in, custom(PAT_PARAM_ATTR_ID,"bytes_is(dwSize)")] void* lpParam
);
204
void_stdcall FuncBytesIs2
(
[out, custom(PAT_PARAM_ATTR_ID, "bytes_is(12)")] void* lpParam
);
205
};

【図１２】


Start
S1
モジュール名/インターフェース名/メソッド名をメモリに保存
S2
呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をHDDに保存
S3
S4
関数定義にバイナリ取得の設定が存在する？
NO
YES
保存するメモリサイズを計算
S5
ポインタ・パラメータの指すメモリの内容を算出したサイズ分HDDに保存
S6
元の関数を呼び出す
S7
リターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をHDDに保存
S8
S9
関数定義にバイナリ取得の設定が存在する？
NO
YES
保存するメモリサイズを計算
S10
ポインタ・パラメータの指すメモリの内容を算出したサイズ分HDDに保存
S11
S12
終了命令？
NO
YES
End
S13

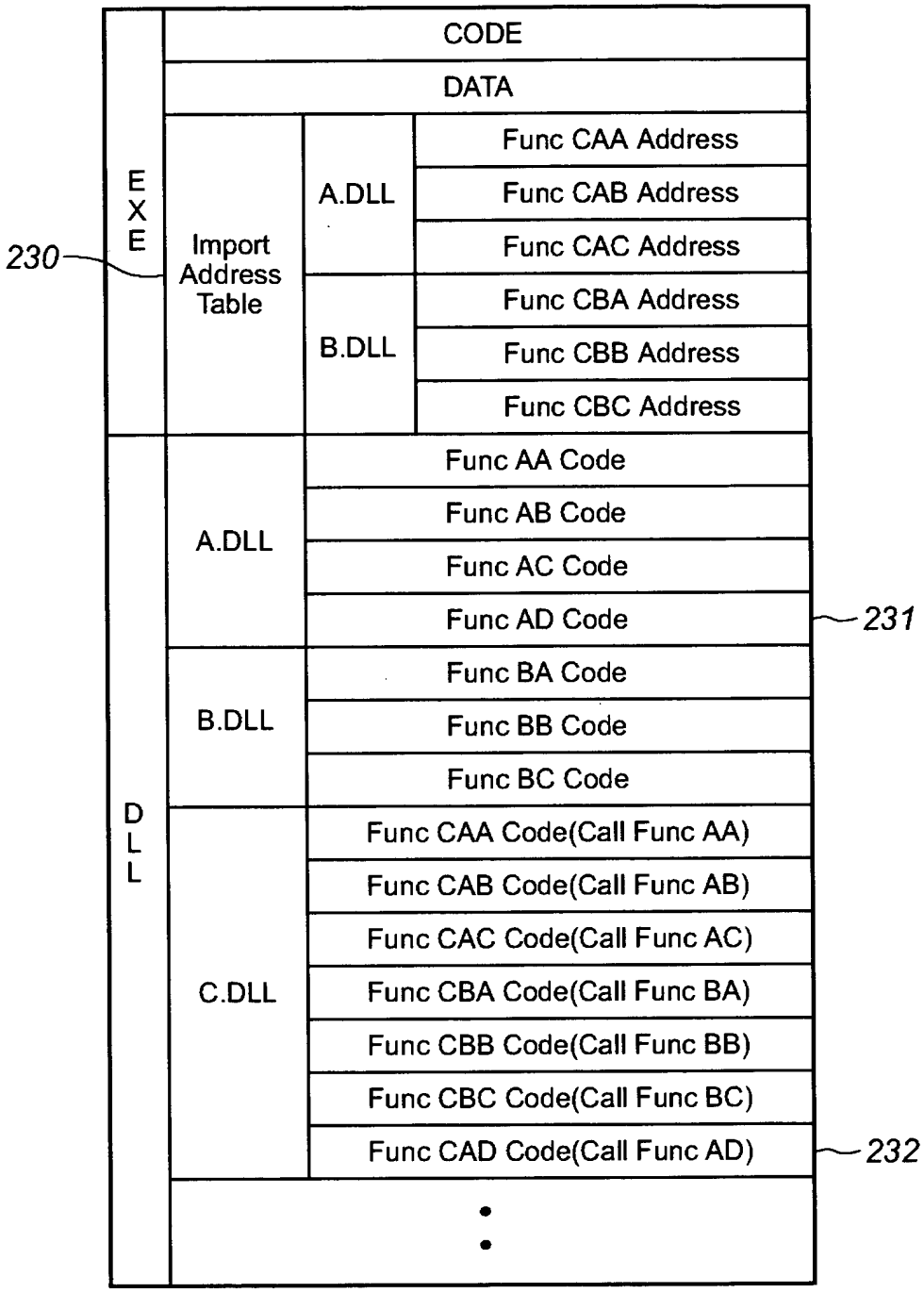
【図１３】


210
モジュール名：TestDllStd. DLL
関数名：FuncBinIdls
引数（in）：
引数（out）：long* IpParam : 0x5034206D/0x10, DataID=0x0001
戻り値：void :
In時刻：2002/03/25 22 : 24 : 12. 025
Out時刻：2002/03/25 22 : 24 : 12. 035
モジュール名：TestDllStd. DLL
関数名：FuncSizeIs
引数（in）：DWORD dwCount : 10
引数（out）：int* IpnParam : 0x5034207D/0x5. DataID=0x0002
戻り値：void :
In時刻：2002/03/25 22 : 24 : 12. 046
Out時刻：2002/03/25 22 : 24 : 12. 057
モジュール名：TestDllStd. DLL
関数名：FuncLengthIs
引数（in）：DWORD dwLength : 5
引数（out）：char *IpszParam : 0x503860C/0x66, DataID=0x0003
戻り値：void :
In時刻：2002/03/25 22 : 24 : 12. 068
Out時刻：2002/03/25 22 : 24 : 12. 079
モジュール名：TestDllStd. DLL
関数名：FuncBytesIs
引数（in）：DWORD dwSize : 7
引数（out）：void* IpParam : 0x503870C/, DataID=0x0004
戻り値：void :
In時刻：2002/03/25 22 : 24 : 12. 100
Out時刻：2002/03/25 22 : 24 : 12. 179
...
211
DataID : 0x0001
Size : 4
00000000 : 10 00 00 00
DataID : 0x0002
Size : 40
00000000 : 05 00 00 00 4A 03 A5 20
00000008 : 06 00 00 00 4B 03 A5 20
00000010 : 07 00 00 00 4C 03 A5 20
00000018 : 08 00 00 00 4D 03 A5 20
00000020 : 09 00 00 00 4E 03 A5 20
DataID : 0x0003
Size : 5
00000000 : 66 4A 70 50 00
DataID : 0x0004
00000000 : 01 5D 66 B2 20 49 20
...

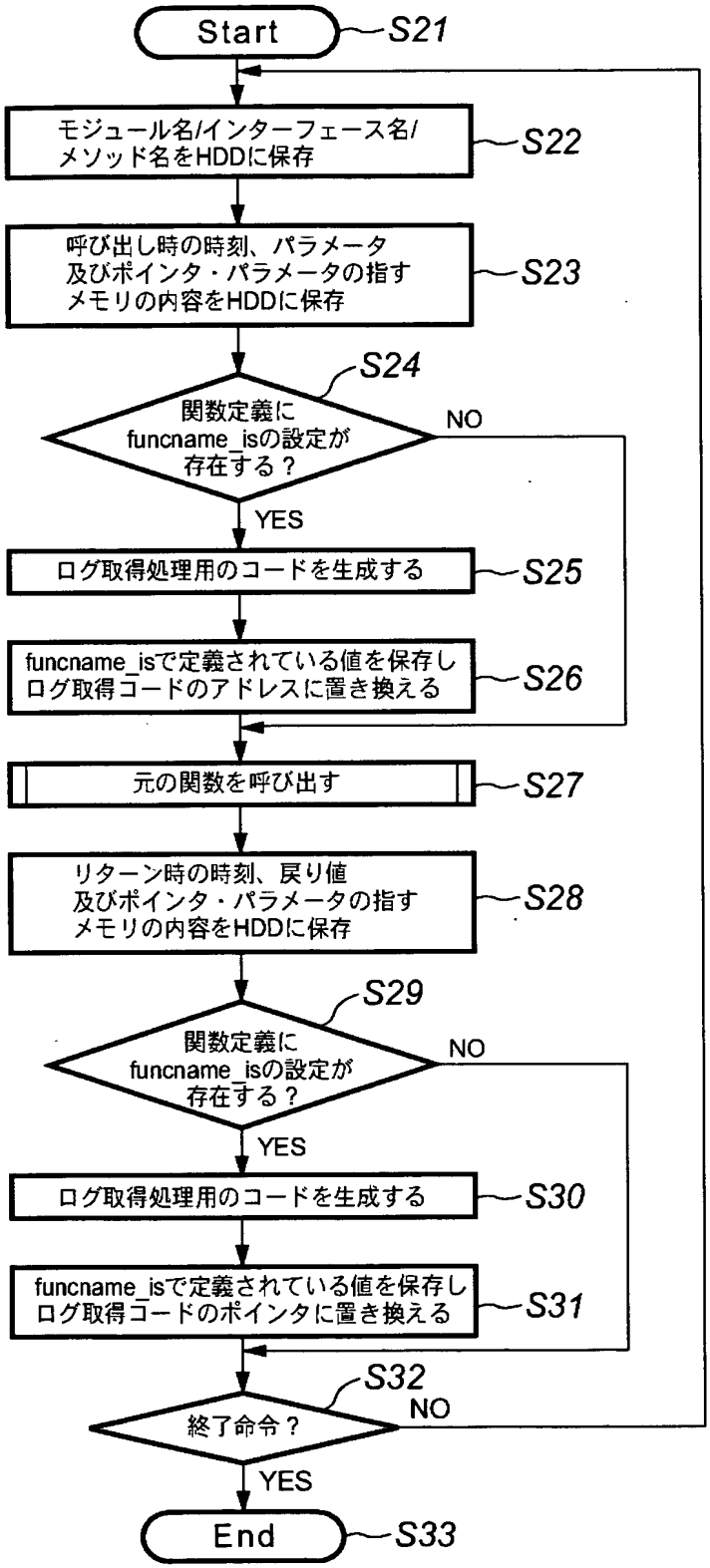
【図１４】

```
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-00000000000

typedef [public] struct ~220
{
        [in, custom(PAT_PARAM_ATTR_ID,"funcname_is(FuncInternal1)")] DWORD pfnFuncInternal1;
        [in, custom(PAT_PARAM_ATTR_ID,"funcname_is(FuncInternal2)")] DWORD pfnFuncInternal2;
        [in, custom(PAT_PARAM_ATTR_ID,"funcname_is(FuncInternal3)")] DWORD pfnFuncInternal3;
        [in, custom(PAT_PARAM_ATTR_ID,"funcname_is(FuncInternal4)")] DWORD pfnFuncInternal4;
}FUNCPOINTERARRAY;

interface
test
{
        void _stdcall SetCallBack                                          ~221
        (
                [in, custom(PAT_PARAM_ATTR_ID,"funcname_is(FuncCallBack)")] DWORD
pfnFuncCallBack
        );
        void FuncCallBack([in] int nParam); ~222

        void _stdcall GetFuncPointer                                       ~223
        (
                [out, custom(PAT_PARAM_ATTR_ID,"funcname_is(FuncInternal)")] DWORD
pfnFuncInternal
        );
        void FuncInternal([in, out] char* lpszParam); ~224

        void _stdcall GetFuncPointerArray
        (
                [out]FUNCPOINTERARRAY* pFuncPointerArray; ~225
        );
        void FuncInternal1([in] int nParam);
        void FuncInternal2([in. out] char* lpzaParam);
        void FuncInternal3([out] DWORD* dwParam); ~226
        void FuncInternal4();
};
```

【図１５】


EXE
DLL
CODE
DATA
Import Address Table
230
A.DLL
B.DLL
Func CAA Address
Func CAB Address
Func CAC Address
Func CBA Address
Func CBB Address
Func CBC Address
A.DLL
B.DLL
C.DLL
Func AA Code
Func AB Code
Func AC Code
Func AD Code
231
Func BA Code
Func BB Code
Func BC Code
Func CAA Code(Call Func AA)
Func CAB Code(Call Func AB)
Func CAC Code(Call Func AC)
Func CBA Code(Call Func BA)
Func CBB Code(Call Func BB)
Func CBC Code(Call Func BC)
Func CAD Code(Call Func AD)
232

【図１６】


Start
S21
モジュール名/インターフェース名/
メソッド名をHDDに保存
S22
呼び出し時の時刻、パラメータ
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S23
S24
関数定義に
funcname_isの設定が
存在する？
NO
YES
ログ取得処理用のコードを生成する
S25
funcname_isで定義されている値を保存し
ログ取得コードのアドレスに置き換える
S26
元の関数を呼び出す
S27
リターン時の時刻、戻り値
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S28
S29
関数定義に
funcname_isの設定が
存在する？
NO
YES
ログ取得処理用のコードを生成する
S30
funcname_isで定義されている値を保存し
ログ取得コードのポインタに置き換える
S31
S32
終了命令？
NO
YES
End
S33

【図１７】

モジュール名：	TestDllStd. DLL
関数名：	FuncSetCallBack
引数（in）：	DWORD pfnFuncCallBack : 0x0299103F
引数（out）：	
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 025
Out時刻：	2002/03/25 22 : 24 : 12. 035
モジュール名：	TestDllStd. DLL
関数名：	GetFuncPointer
引数（in）：	
引数（out）：	DWORD pfnFuncInternal : 0x029913dF
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 046
Out時刻：	2002/03/25 22 : 24 : 12. 057
モジュール名：	TestDllStd. DLL
関数名：	GetFuncPointerArray
引数（in）：	
引数（out）：	FUNCPOINTERARRAY* pFuncPointerArray : 0x503860C
	DWORD FUNCPOINTERARRAY. pfnFuncInternal1 : 0x02997670
	DWORD FUNCPOINTERARRAY. pfnFuncInternal2 : 0x02997708
	DWORD FUNCPOINTERARRAY. pfnFuncInternal3 : 0x029977BE
	DWORD FUNCPOINTERARRAY. pfnFuncInternal4 : 0x0299784F
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 068
Out時刻：	2002/03/25 22 : 24 : 12. 079

【図１８】

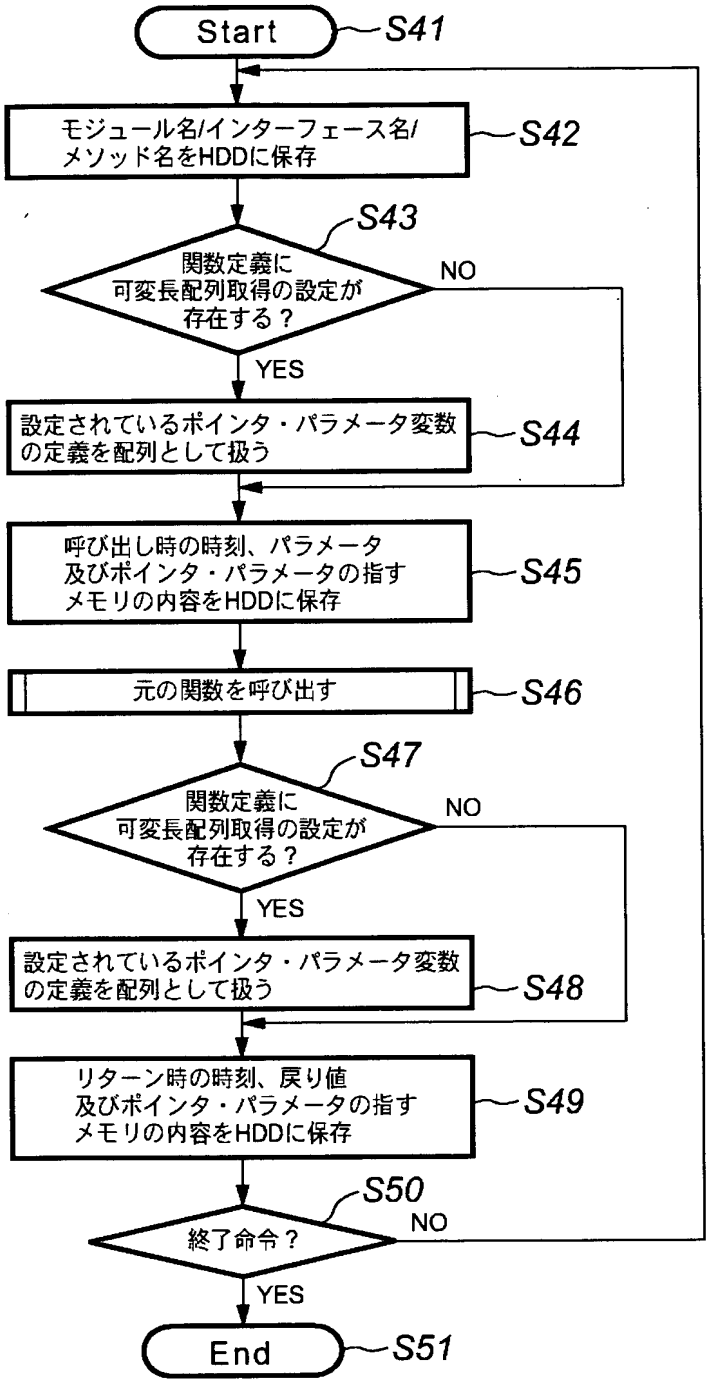
モジュール名：	TestDllStd. DLL
関数名：	FuncSetCallBack
引数（in）：	DWORD pfnFuncCallBack : 0x0299103F
引数（out）：	
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 025
Out時刻：	2002/03/25 22 : 24 : 12. 035
モジュール名：	TestDllStd. DLL
関数名：	FuncCallBack
引数（in）：	int nParam : 100
引数（out）：	
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 036
Out時刻：	2002/03/25 22 : 24 : 12. 040
モジュール名：	TestDllStd. DLL
関数名：	GetFuncPointer
引数（in）：	
引数（out）：	DWORD pfnFuncInternal : 0x029913dF
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 046
Out時刻：	2002/03/25 22 : 24 : 12. 057
モジュール名：	TestDllStd. DLL
関数名：	FuncInternal
引数（in）：	char* lpszParam : 0x5038600/0
引数（out）：	char* lpszParam : 0x5038600/-12
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 060
Out時刻：	2002/03/25 22 : 24 : 12. 065
モジュール名：	TestDllStd. DLL
関数名：	GetFuncPointArray
引数（in）：	
引数（out）：	FUNCPOINTERARRAY* pFuncPointerArray : 0x503860C DWORD FUNCPOINTERARRAY. pfnFuncInternal1 : 0x02997670 DWORD FUNCPOINTERARRAY. pfnFuncInternal2 : 0x02997708 DWORD FUNCPOINTERARRAY. pfnFuncInternal3 : 0x029977BE DWORD FUNCPOINTERARRAY. pfnFuncInternal4 : 0x0299784F
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 068
Out時刻：	2002/03/25 22 : 24 : 12. 079
モジュール名：	TestDllStd. DLL
関数名：	FuncInternal4
引数（in）：	
引数（out）：	
戻り値：	void :
In時刻：	2002/03/25 22 : 24 : 12. 080
Out時刻：	2002/03/25 22 : 24 : 12. 099

【図１９】

```
#define PAT PARAM.ATTR_ID 00000000-0000-0000-0000-000000000000

interface
test
{       void_stdcall FuncArrayIs
        (
                [in] DWORD dwCount,
                [in, out, custom(PAT_PARAM_ATTR_ID,"array_is(dwCount)")] int* lpnParam
        );
};
```

240

【図２０】


Start
S41
モジュール名/インターフェース名/
メソッド名をHDDに保存
S42
S43
関数定義に
可変長配列取得の設定が
存在する？
NO
YES
設定されているポインタ・パラメータ変数
の定義を配列として扱う
S44
呼び出し時の時刻、パラメータ
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S45
元の関数を呼び出す
S46
S47
関数定義に
可変長配列取得の設定が
存在する？
NO
YES
設定されているポインタ・パラメータ変数
の定義を配列として扱う
S48
リターン時の時刻、戻り値
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S49
S50
終了命令？
NO
YES
End
S51

【図２１】

250

```
モジュール名：  TestDllStd. DLL
関数名：        FuncArrayIs
引数 ( in ) :   DWORD dwCount : 4
                int* lpnParam : 0x5034206D/0x00
引数 ( out ) :  int* lpnParam : 0x5034206D/0x01
戻り値：        void :
In時刻：        2002/03/25 22 : 24 : 12. 025
Out時刻：       2002/03/25 22 : 24 : 12. 035

モジュール名：  TestDllStd. DLL
関数名：        FuncArrayIs
引数 ( in ) :   DWORD dwCount : 3
                int* lpnParam : 0x5034207D/0x00
引数 ( out ) :  int* lpnParam : 0x5034207D/0x05
戻り値：        void :
In時刻：        2002/03/25 22 : 24 : 12. 046
Out時刻：       2002/03/25 22 : 24 : 12. 057

...
```

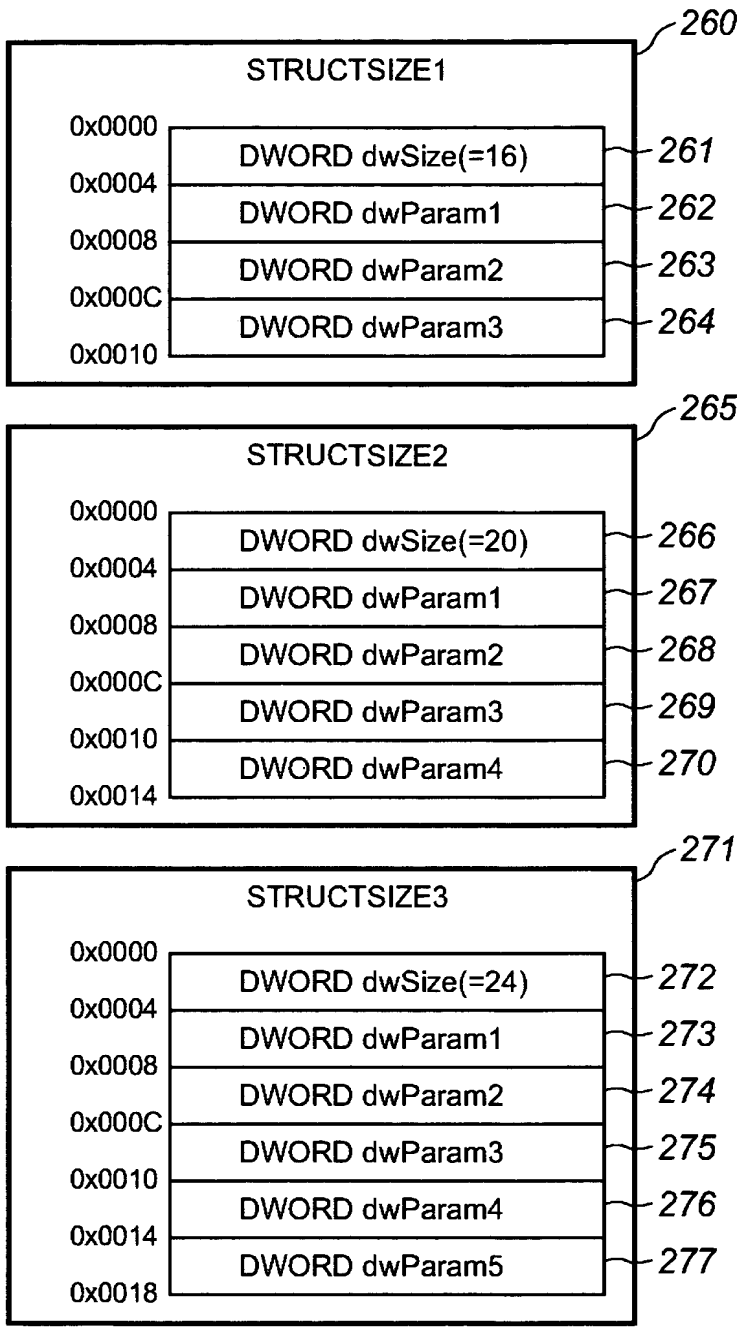
251

```
モジュール名：  TestDllStd. DLL
関数名：        FuncArrayIs
引数 ( in ) :   DWORD dwCount : 4
                int* lpnParam : 0x5034206D/Array ( int : 0 : 0x00, int : 1 : 0x00, int : 2 : 0x00, int : 3 : 0x00 )
引数 ( out ) :  int* lpnParam : 0x5034206D/Array ( int : 0 : 0x01, int : 1 : 0x02, int : 2 : 0x03, int : 3 : 0x04 )
戻り値：        void :
In時刻：        2002/03/25 22 : 24 : 12. 025
Out時刻：       2002/03/25 22 : 24 : 12. 035

モジュール名：  TestDllStd. DLL
関数名：        FuncArrayIs
引数 ( in ) :   DWORD dwCount : 3
                int* lpnParam : 0x5034207D/Array ( int : 0 : 0x00, int : 1 : 0x00, int : 2 : 0x00 )
引数 ( out ) :  int* lpnParam : 0x5034207D/Array ( int : 0 : 0x05, int : 1 : 0x10, int : 2 : 0x15 )
戻り値：        void :
In時刻：        2002/03/25 22 : 24 : 12. 046
Out時刻：       2002/03/25 22 : 24 : 12. 057

...
```

【図２２】

```
typedef struct
{
        DWORD dwSize;
        DWORD dwParam1;
        DWORD dwParam2;
        DWORD dwParam3;
}STRUCTSIZE1;

typedef struct
{       DWORD dwSize;
        DWORD dwParam1;
        DWORD dwParam2;
        DWORD dwParam3;
        DWORD dwParam4;
}STRUCTSIZE2;

typedef struct
{       DWORD dwSize;
        DWORD dwParam1;
        DWORD dwParam2;
        DWORD dwParam3;
        DWORD dwParam4;
        DWORD dwParam5;
}STRUCTSIZE3;

void FuncGetData (DWORD dwKind, void* lpBuf)
{
        switch(dwKind)
        {
        case 1:
                //lpBufをSTRUCTSIZE1のポインタとして処理
                break;
        case 2:
                //lpBufをSTRUCTSIZE2のポインタとして処理
                break;
        case 3:
                //lpBufをSTRUCTSIZE3のポインタとして処理
                break;
        }
}
```

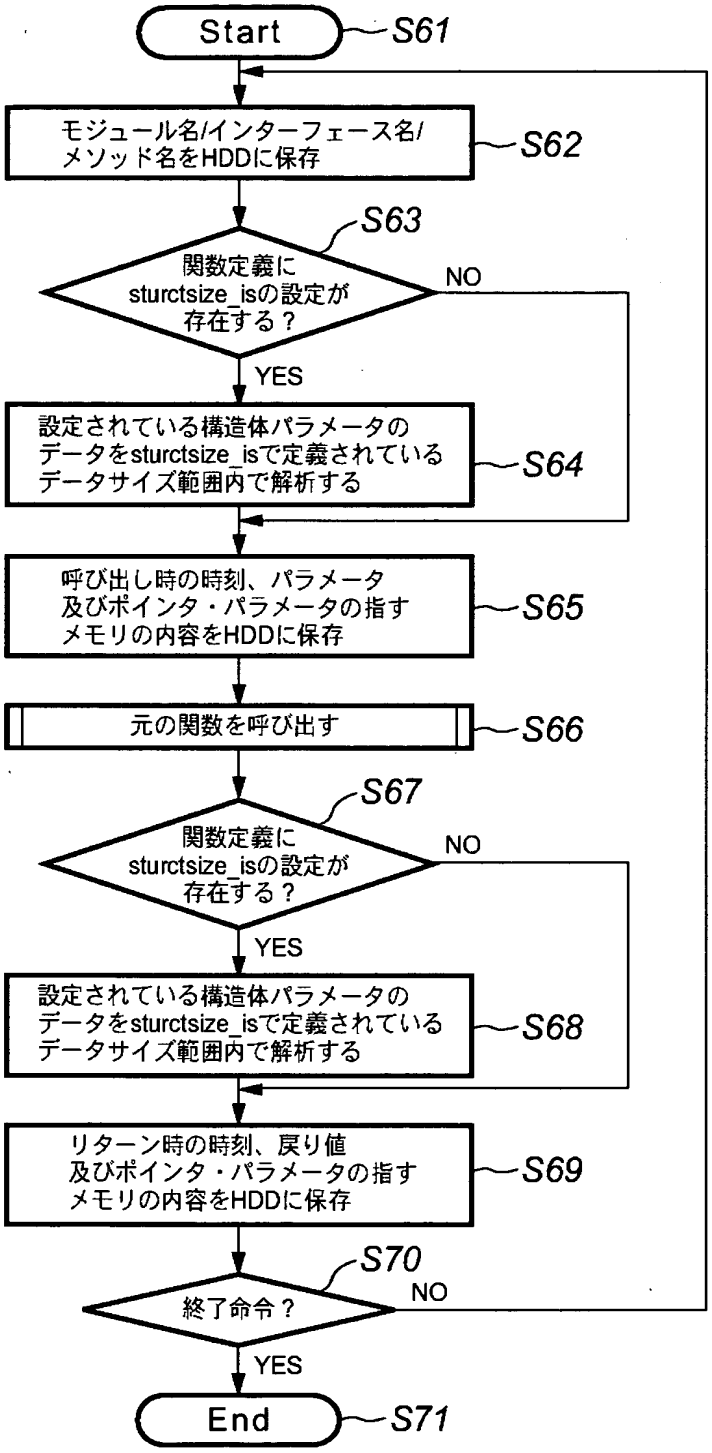
【図２３】


260
STRUCTSIZE1
0x0000
DWORD dwSize(=16)
261
0x0004
DWORD dwParam1
262
0x0008
DWORD dwParam2
263
0x000C
DWORD dwParam3
264
0x0010
265
STRUCTSIZE2
0x0000
DWORD dwSize(=20)
266
0x0004
DWORD dwParam1
267
0x0008
DWORD dwParam2
268
0x000C
DWORD dwParam3
269
0x0010
DWORD dwParam4
270
0x0014
271
STRUCTSIZE3
0x0000
DWORD dwSize(=24)
272
0x0004
DWORD dwParam1
273
0x0008
DWORD dwParam2
274
0x000C
DWORD dwParam3
275
0x0010
DWORD dwParam4
276
0x0014
DWORD dwParam5
277
0x0018

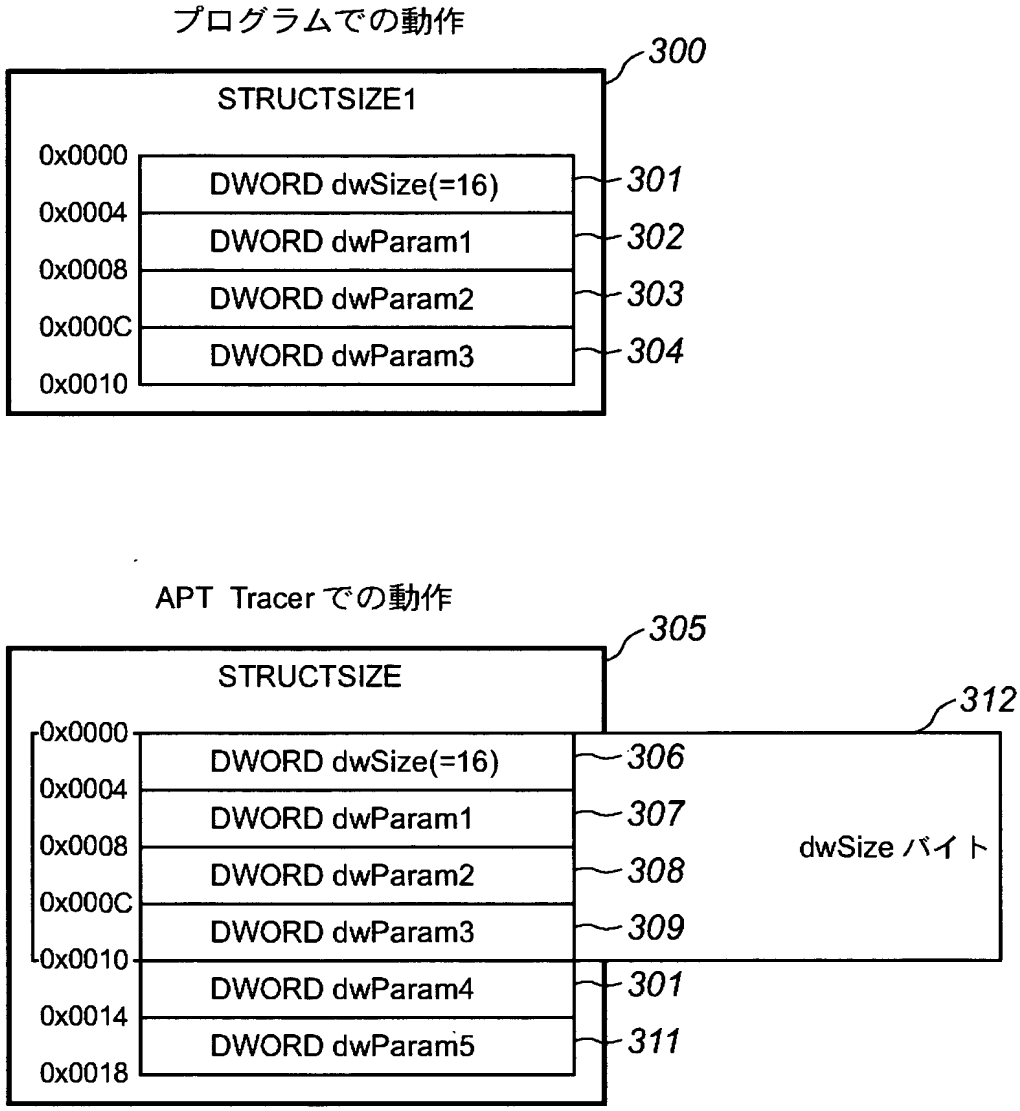
【図２４】

```
#define PAT PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000

typedef [public] struct                                          290
{
        [custom (PAT_PARAM_ATTR_ID,"structsize_is()")]DWORD dwSize;
        DWORD dwParam1 ;
        DWORD dwParam2 ;
        DWORD dwParam3 ;
        DWORD dwParam4 ;
        DWORD dwParam5 ;
}STRUCTSIZE;                                                     291

interface
test
{
        void FuncGetData
        (
                [in] DWORD dwKind,                               292
                [in, out] STRUCTSIZE* lpBuf
        );

};
```

【図２５】


Start
S61
モジュール名/インターフェース名/
メソッド名をHDDに保存
S62
S63
関数定義に
sturctsize_isの設定が
存在する？
NO
YES
設定されている構造体パラメータの
データをsturctsize_isで定義されている
データサイズ範囲内で解析する
S64
呼び出し時の時刻、パラメータ
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S65
元の関数を呼び出す
S66
S67
関数定義に
sturctsize_isの設定が
存在する？
NO
YES
設定されている構造体パラメータの
データをsturctsize_isで定義されている
データサイズ範囲内で解析する
S68
リターン時の時刻、戻り値
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S69
S70
終了命令？
NO
YES
End
S71

【図２６】


プログラムでの動作
300
STRUCTSIZE1
0x0000
DWORD dwSize(=16)
301
0x0004
DWORD dwParam1
302
0x0008
DWORD dwParam2
303
0x000C
DWORD dwParam3
304
0x0010
APT Tracer での動作
305
STRUCTSIZE
312
0x0000
DWORD dwSize(=16)
306
0x0004
DWORD dwParam1
307
0x0008
DWORD dwParam2
308
0x000C
DWORD dwParam3
309
dwSize バイト
0x0010
DWORD dwParam4
301
0x0014
DWORD dwParam5
311
0x0018

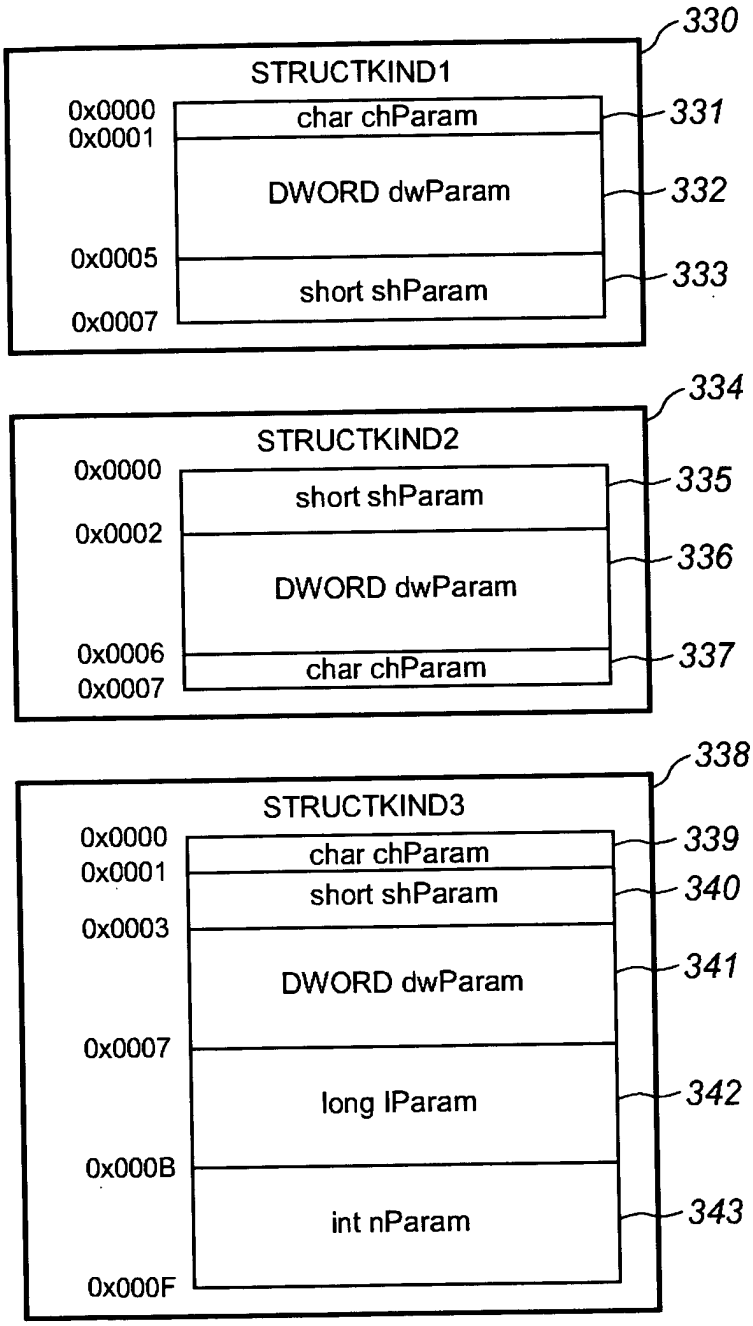
【図２７】

```
モジュール名：　TestDllStd. DLL
関数名：　FuncGetData
引数 ( in ) :　DWORD dwKind : 1
　　STRUCTSIZE* pBuf : 0x503860C
　　DWORD STRUCTSIZE. dwSize : 16
　　DWORD STRUCTSIZE. dwParam1 : 0
　　DWORD STRUCTSIZE. dwParam2 : 0
　　DWORD STRUCTSIZE. dwParam3 : 0
引数 ( out ) :　STRUCTSIZE* pBuf : 0x503860C
　　DWORD STRUCTSIZE. dwSize : 16
　　DWORD STRUCTSIZE. dwParam1 : 1
　　DWORD STRUCTSIZE. dwParam2 : 2
　　DWORD STRUCTSIZE. dwParam3 : 3
戻り値：　void :
In時刻：　2002/03/25 22 : 24 : 12. 025
Out時刻：　2002/03/25 22 : 24 : 12. 035

モジュール名：　TestDllStd. DLL
関数名：　FuncGetData
引数 ( in ) :　DWORD dwKind : 3
　　STRUCTSIZE* pBuf : 0x503990C
　　DWORD STRUCTSIZE. dwSize : 24
　　DWORD STRUCTSIZE. dwParam1 : 0
　　DWORD STRUCTSIZE. dwParam2 : 0
　　DWORD STRUCTSIZE. dwParam3 : 0
　　DWORD STRUCTSIZE. dwParam4 : 0
　　DWORD STRUCTSIZE. dwParam5 : 0
引数 ( out ) :　STRUCTSIZE* pBuf : 0x503990C
　　DWORD STRUCTSIZE. dwSize : 24
　　DWORD STRUCTSIZE. dwParam1 : 10
　　DWORD STRUCTSIZE. dwParam2 : 20
　　DWORD STRUCTSIZE. dwParam3 : 30
　　DWORD STRUCTSIZE. dwParam4 : 40
　　DWORD STRUCTSIZE. dwParam5 : 50
戻り値：　void :
In時刻：　2002/03/25 22 : 24 : 12. 046
Out時刻：　2002/03/25 22 : 24 : 12. 057

・・・
```

【図２８】

```
typedef struct
{
        char chParam;
        DWORD dwParam;
        short shParam;
}STRUCTKIND1;

typedef struct
{
        short shParam ;
        DWORD dwParam;
        char chParam ;
}STRUCTKIND2;

typedef struct
{
        char chParam ;
        DWORD dwParam ;
        short shParam ;
        long lParam ;
        int nParam ;
}STRUCTKIND3;

void FuncGetData (DWORD dwKind, void* lpBuf)
{

        switch(dwKind)
        {
        case 1:
                //lpBufをSTRUCTKIND1のポインタとして処理
                break;
        case 2:
                //lpBufをSTRUCTKIND2のポインタとして処理
                break;
        case 3:
                //lpBufをSTRUCTKIND3のポインタとして処理
                break;
        }
}
```

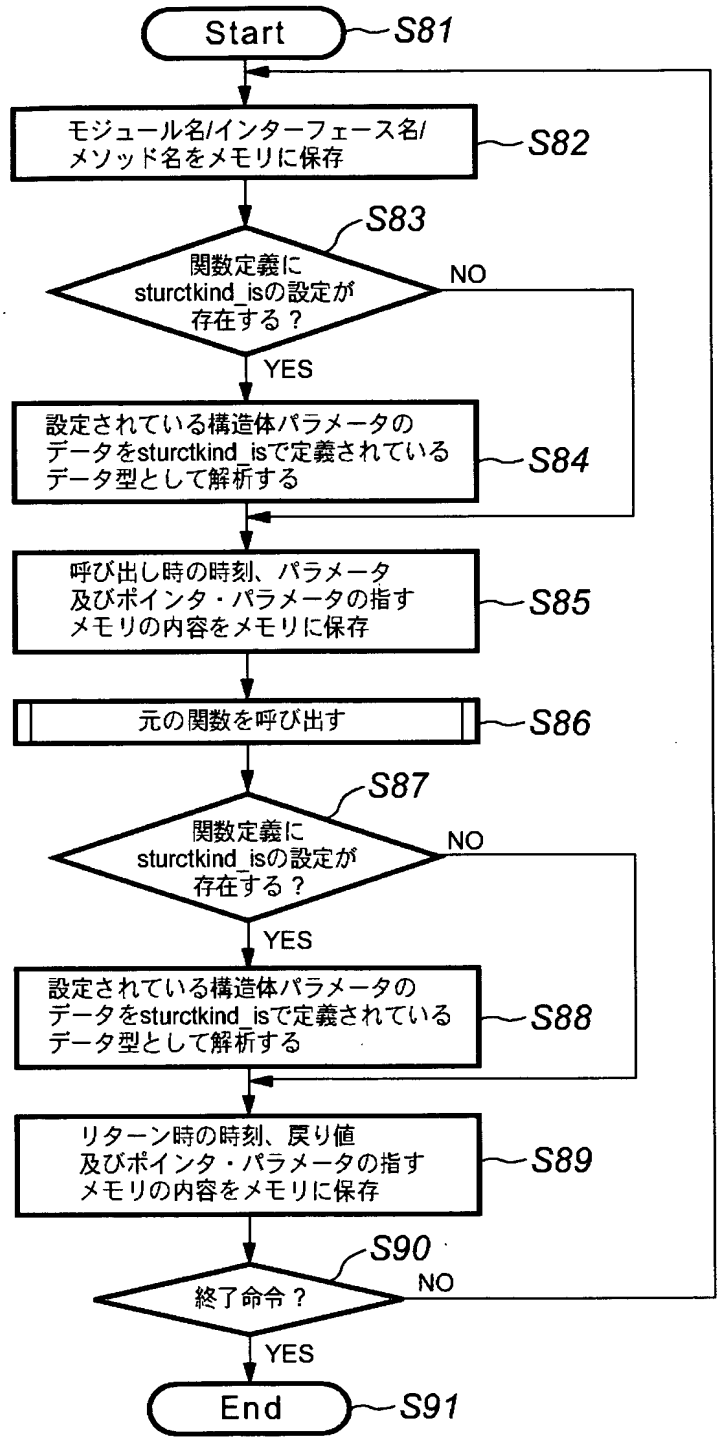
【図２９】


330
STRUCTKIND1
0x0000
0x0001
char chParam
331
DWORD dwParam
332
0x0005
short shParam
333
0x0007
334
STRUCTKIND2
0x0000
short shParam
335
0x0002
DWORD dwParam
336
0x0006
0x0007
char chParam
337
338
STRUCTKIND3
0x0000
0x0001
char chParam
339
short shParam
340
0x0003
DWORD dwParam
341
0x0007
long lParam
342
0x000B
int nParam
343
0x000F

【図３０】

```
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-00000000000

typedef [public] struct
{
        char chParam;
        DWORD dwParam;
        short shParam;
}STRUCTKIND1;

typedef  [public]  struct
{
        short shParam;
        DWORD dwParam;
        char chParam;
}STRUCTKIND2;

typedef  [public]   struct
{
        char chParam;
        short shParam;
        DWORD dwParam;
        long lParam;
        int  nParam;
}STRUCTKIDN3;

interface
test
{
        void FuncGetData
        (
                [in]DWORD dwKind,
                [in, out, custom(PAT_PARAM_ATTR_ID,
                "structKind_is (dwKind : 1: STRUCTKIND1*, 2: STRUCTKIND2*, 3: STRUCTKIND3*  )")]
                void* lpBuf
        );
};
```

【図３１】


Start
S81
モジュール名/インターフェース名/
メソッド名をメモリに保存
S82
S83
関数定義に
sturctkind_isの設定が
存在する？
NO
YES
設定されている構造体パラメータの
データをsturctkind_isで定義されている
データ型として解析する
S84
呼び出し時の時刻、パラメータ
及びポインタ・パラメータの指す
メモリの内容をメモリに保存
S85
元の関数を呼び出す
S86
S87
関数定義に
sturctkind_isの設定が
存在する？
NO
YES
設定されている構造体パラメータの
データをsturctkind_isで定義されている
データ型として解析する
S88
リターン時の時刻、戻り値
及びポインタ・パラメータの指す
メモリの内容をメモリに保存
S89
S90
終了命令？
NO
YES
End
S91

【図３２】

モジュール名：　TestDllStd. DLL
関数名：　FuncGetData
引数（in）：　DWORD dwKind : 1
STRUCTKIND1* pBuf : 0x503860C
char STRUCTKIND1. chParam : 0
DWORD STRUCTKIND1. dwParam : 0
short STRUCTKIND1. shParam : 0
引数（out）：　STRUCTKIND1* pBuf : 0x503860C
char STRUCTKIND1. chParam : 1
DWORD STRUCTKIND1. dwParam : 2
short STRUCTKIND1. shParam : 3

戻り値：　void :
In時刻：　2002/03/25 22 : 24 : 12. 025
Out時刻：　2002/03/25 22 : 24 : 12. 035

モジュール名：　TestDllStd. DLL
関数名：　FuncGetData
引数（in）：　DWORD dwKind : 3
STRUCTKIND3* pBuf : 0x503990C
char STRUCTKIND3. chParam : 0
short STRUCTKIND3. shParam : 0
DWORD STRUCTKIND3. dwParam : 0
long STRUCTKIND3. lParam : 0
int STRUCTKIND3. nParam : 0
引数（out）：　STRUCTKIND3* pBuf : 0x503990C
char STRUCTKIND3. chParam : 10
short STRUCTKIND3. shParam : 20
DWORD STRUCTKIND3. dwParam : 30
long STRUCTKIND3. lParam : 40
int STRUCTKIND3. nParam : 50
戻り値：　void :
In時刻：　2002/03/25 22 : 24 : 12. 046
Out時刻：　2002/03/25 22 : 24 : 12. 057

・・・

【書類名】　要約書

【要約】

【課題】　ソフトウェアの処理ログを容易に取得でき、かつ、バグの解析のための工数を削減することが可能なログ取得方法を提供する。

【解決手段】　関数（FuncAA）を備えるプログラム９１において、実行中のログを取得するログ取得方法であって、ロードされた関数（FuncAA）のアドレスを、ログ取得のための関数（９２）のアドレスに書き換える工程を備え、ログ取得のための関数（９２）は、関数（FuncAA）のアドレスを呼び出し（９６）、該所定の処理を実行させ（９７）、受け取った実行結果（９８）をプログラム９１に渡す工程（１０１）と、プログラム９１内の関数定義に、ポインタ・パラメータが所定の定義方法で定義されているか否かを判断する工程と、定義されていた場合、該定義方法に基づいて該ポインタ・パラメータの指すメモリの内容をログとして記録する工程とを備える。

【選択図】　図４Ａ

特願２００２－１９１１２８

出　願　人　履　歴　情　報

識別番号　　［０００００１００７］

１．変更年月日　　１９９０年　８月３０日
［変更理由］　　新規登録
住　所　　東京都大田区下丸子３丁目３０番２号
氏　名　　キヤノン株式会社